Data Transfer Object (DTO)

Topics

- Data Transfer Object (DTO)
  - Providing a list of objects to the UI Layer
  - Sending data from the UI do the Domain Layer
- Mappers

DTO Pattern

- **Problem**
  - How to carry data from one element of the system (e.g. layer, component) to another, **reducing coupling** and/or meeting other qualities/needs (such as hiding some attributes)?

- **Solution**
  - Create a **Data Transfer Object (DTO)** holding all the data required to fulfill the intended purpose
  - Typically, a third object (called a **Mapper** or Assembler) is used to convert the original data (e.g. objects from the Domain Layer) to the DTO and vice-versa
  - The DTO must not have any business logic or behavior
  - The DTO is just a **bag of data** and therefore its attributes can pe public and no *getters* and *setters* are needed

Applying DTO

- From the Domain Layer to the UI Layer (UI ← Domain)
  - When **providing a list of objects** for the user to select one (or more)
  - When **providing a report** whose content aggregates and/or summarizes data spread across several domain objects

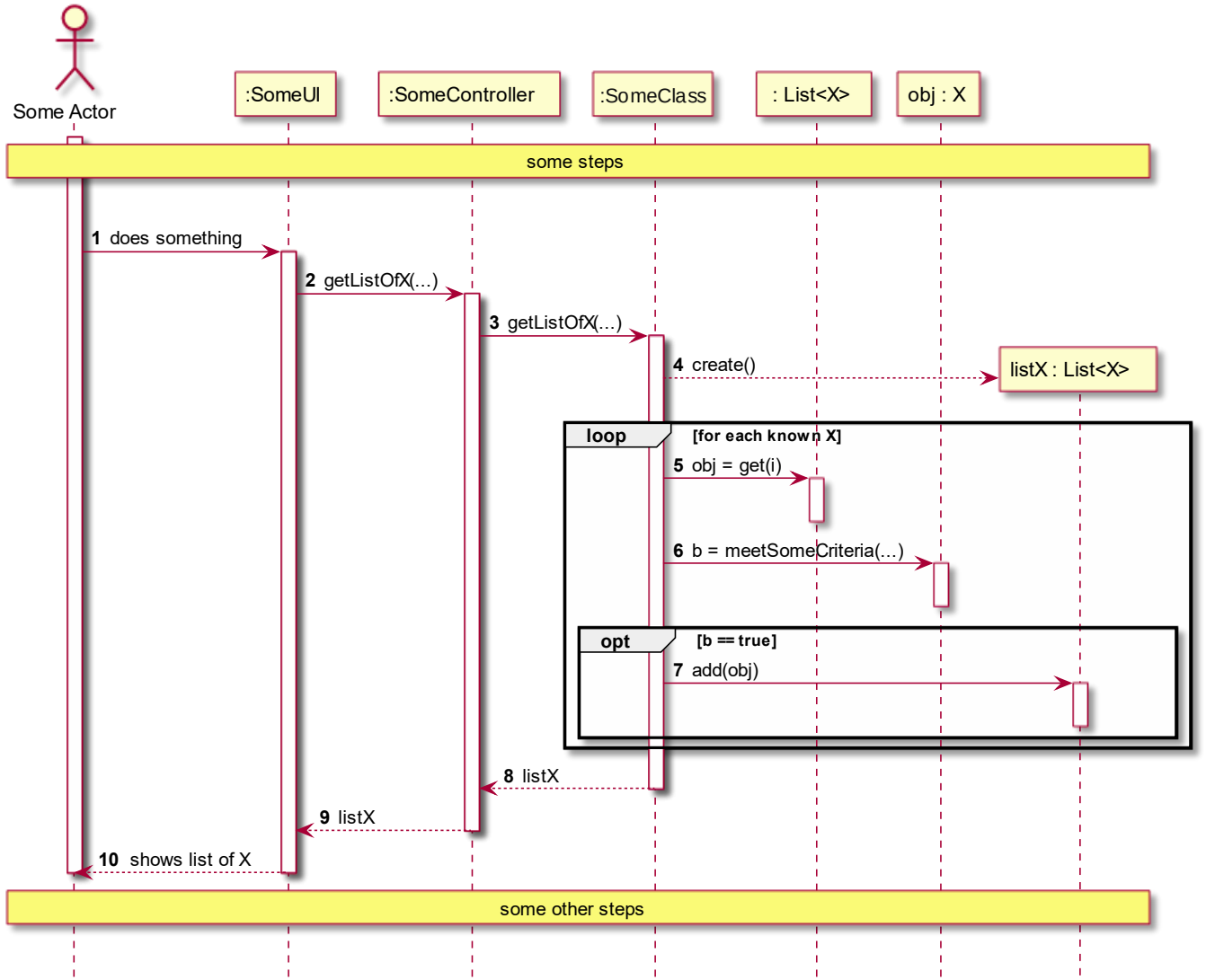
- From the UI Layer to the Domain Layer (UI → Domain)
  - When **sending user-inputted data to the domain** and the controller's method takes more than four or five parameters (e.g. creating a new client with several attributes)

- From the Domain Layer to a Persistence Layer
  - When dealing with the requirement: "The application should use object serialization to ensure data persistence between two runs of the application."
  - *Note: For now, do not consider this while designing the system*

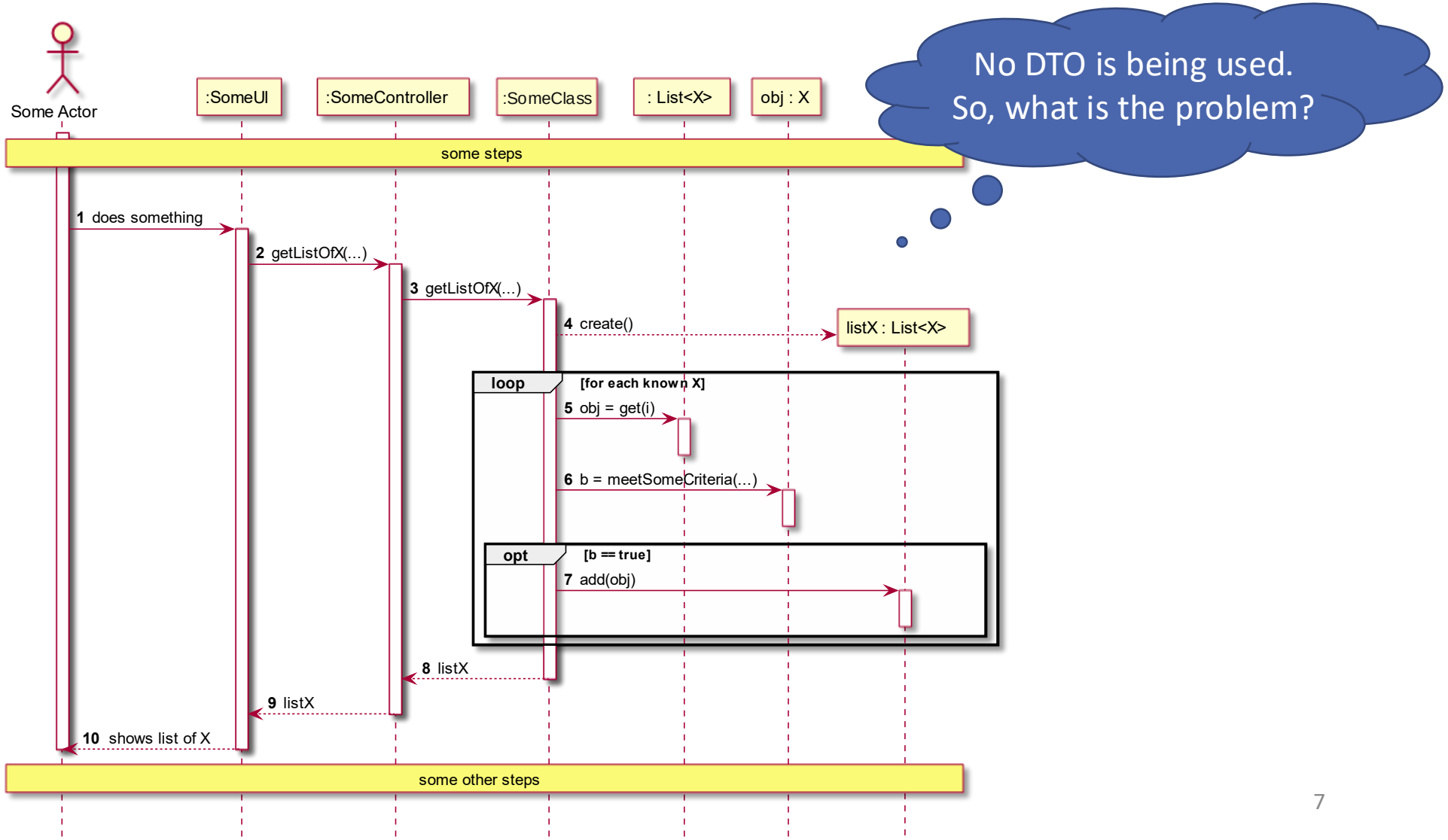
From the Domain Layer to the UI Layer

Providing a list of objects for the user to select one (or more)

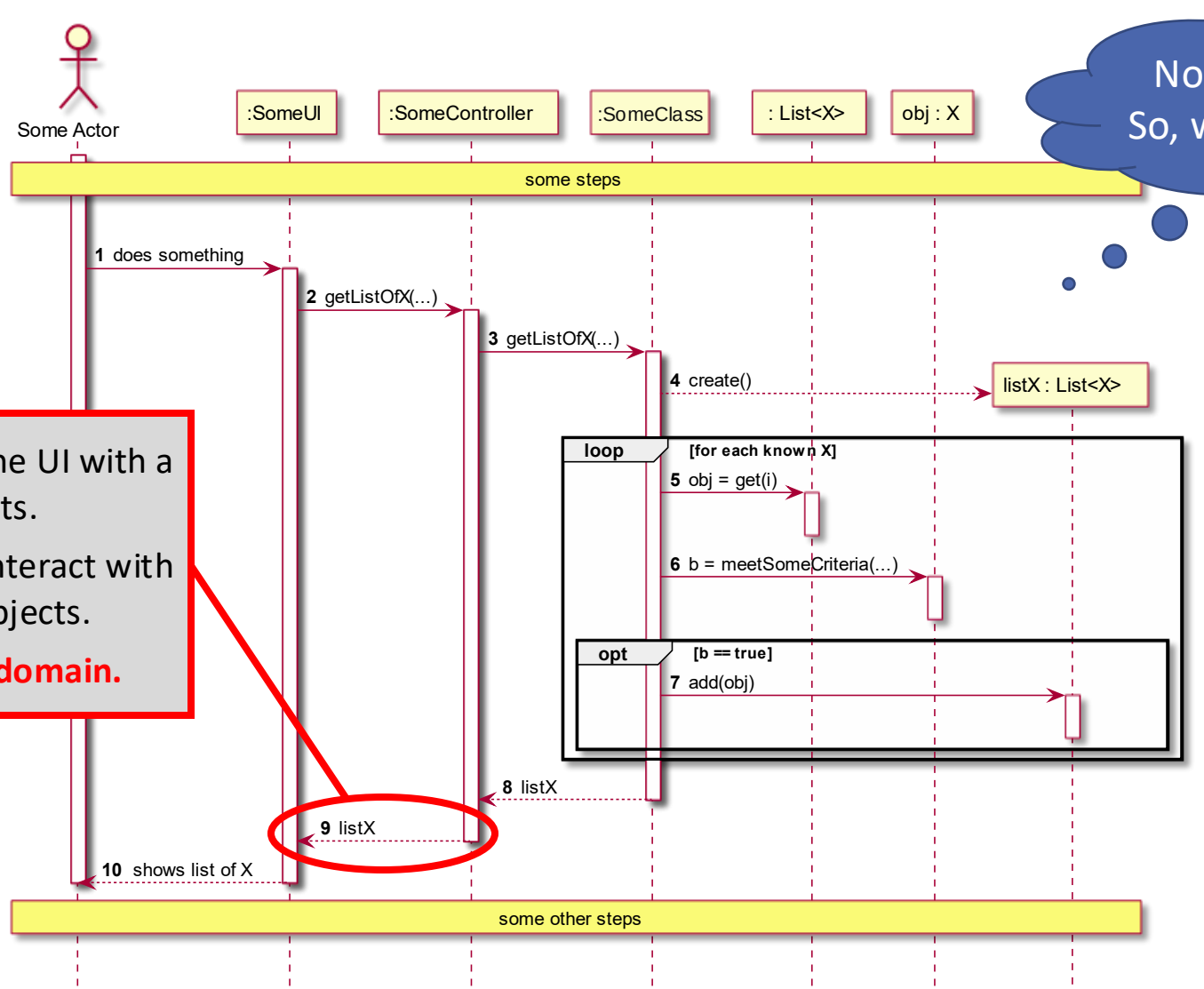
UI ← Domain: Providing a List of Domain Objects (1/4)



UI ← Domain: Providing a List of Domain Objects (2/4)



UI ← Domain: Providing a List of Domain Objects (3/4)



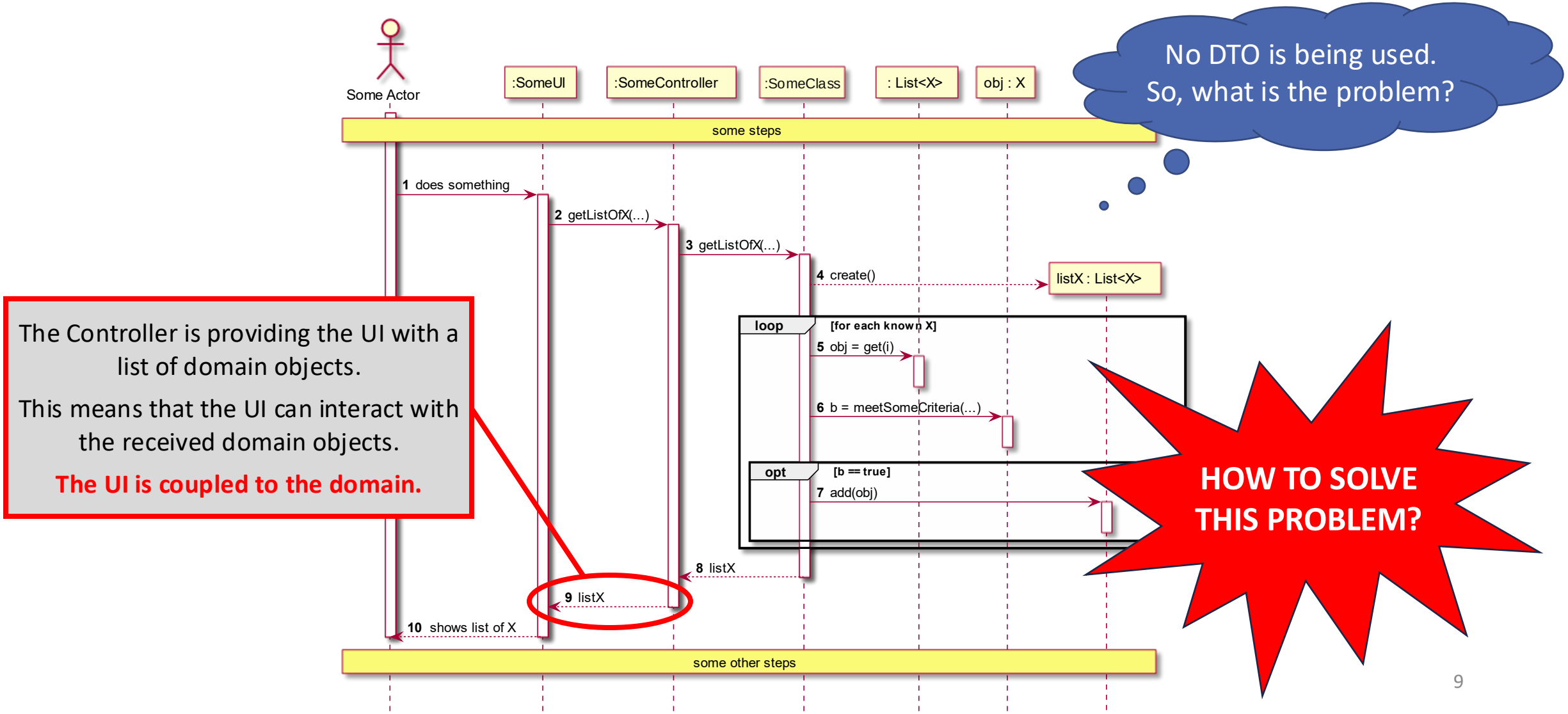
No DTO is being used.
So, what is the problem?

The Controller is providing the UI with a list of domain objects.

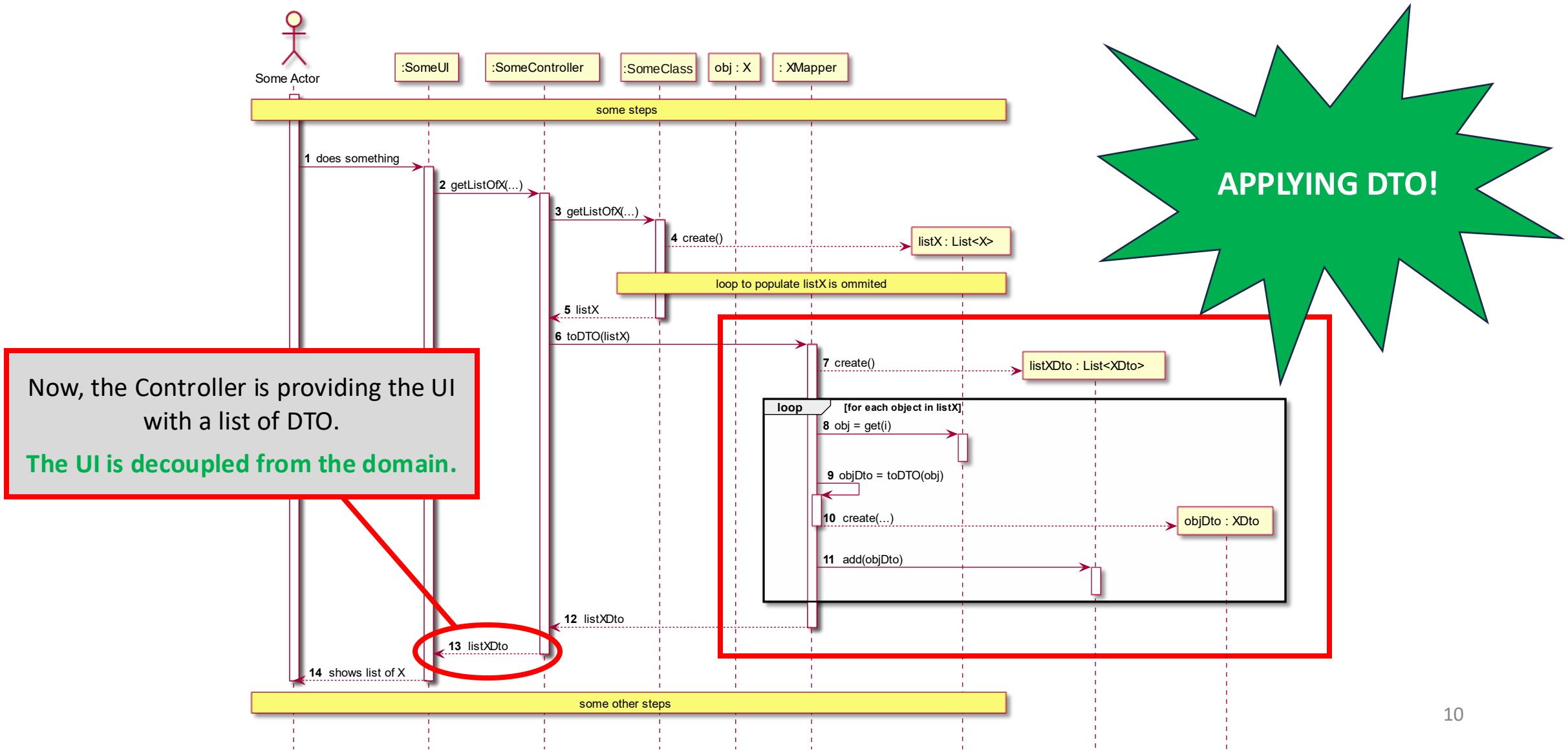
This means that the UI can interact with the received domain objects.

The UI is coupled to the domain.

UI ← Domain: Providing a List of Domain Objects (4/4)

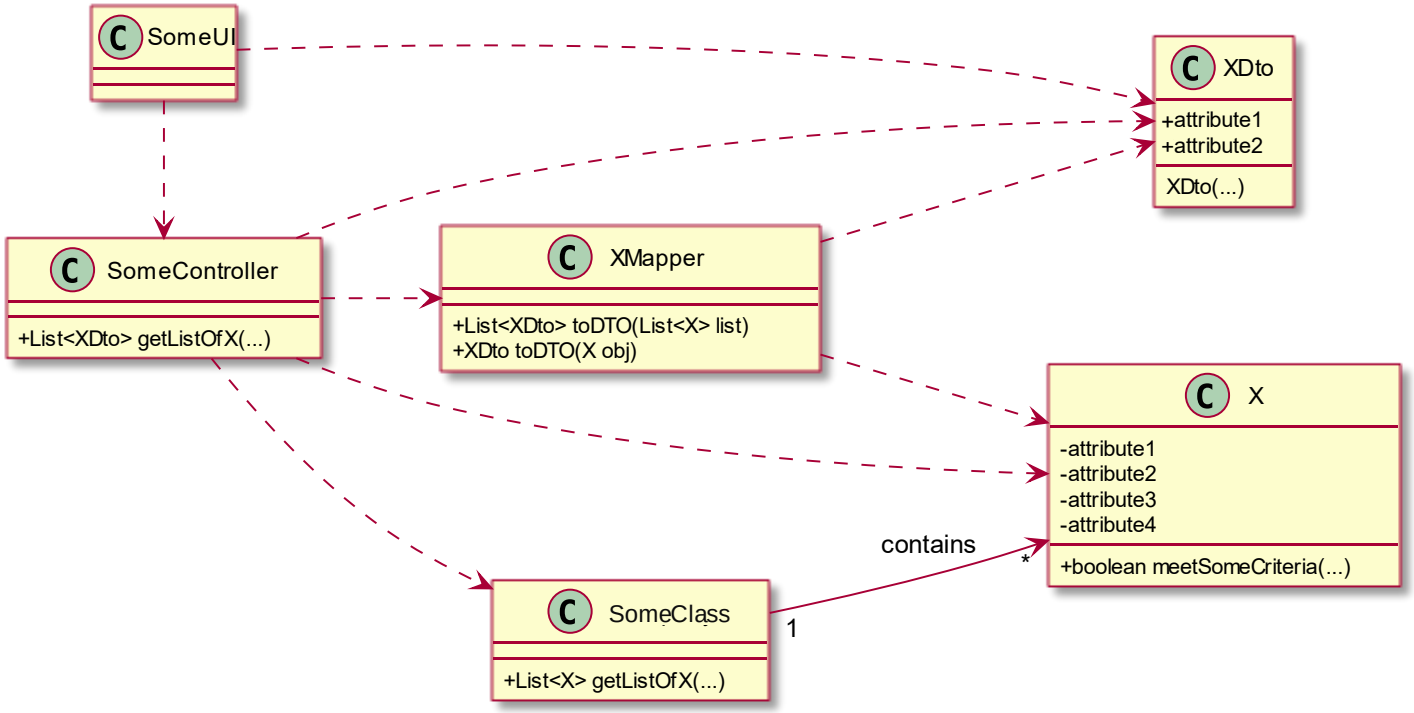


UI ← Domain: Providing a List of DTO (1/2)



UI ← Domain: Providing a List of DTO (2/2)

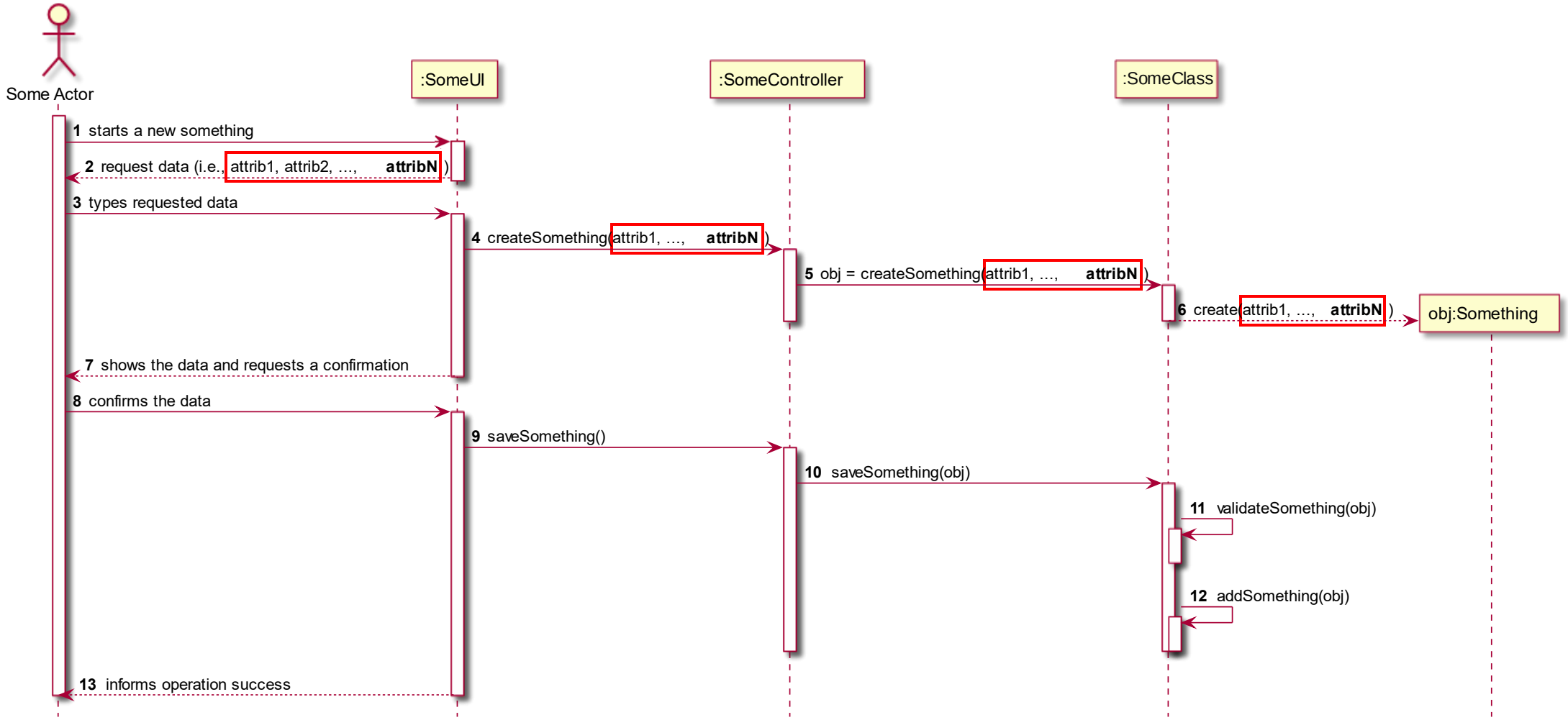
- The **Controller acts as an intermediary** between the UI and the domain
- The Controller does not have the responsibility to process data, but knows who does – it **tells the Mapper to process the data** (e.g. convert to DTO)



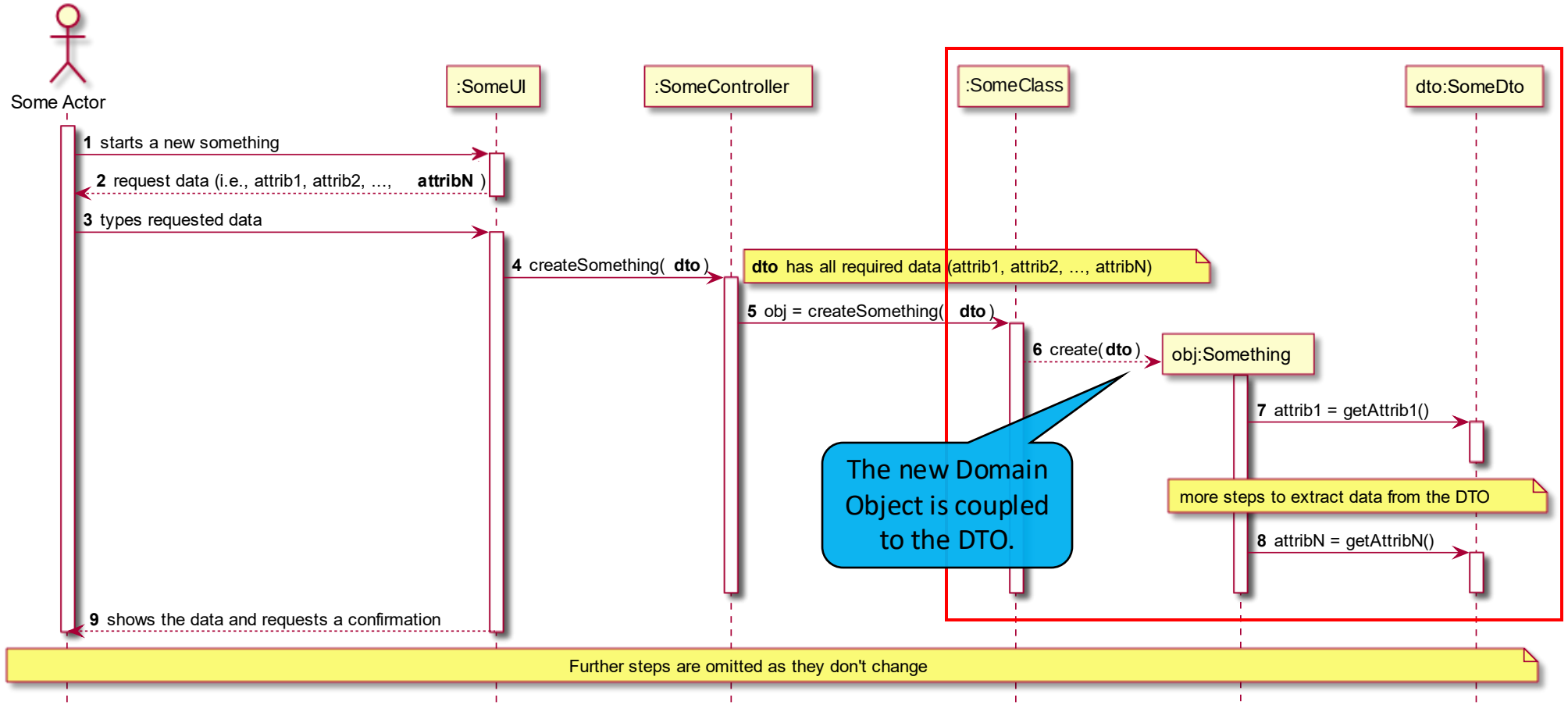
From the UI Layer to the Domain Layer

Sending user-inputted data to the domain

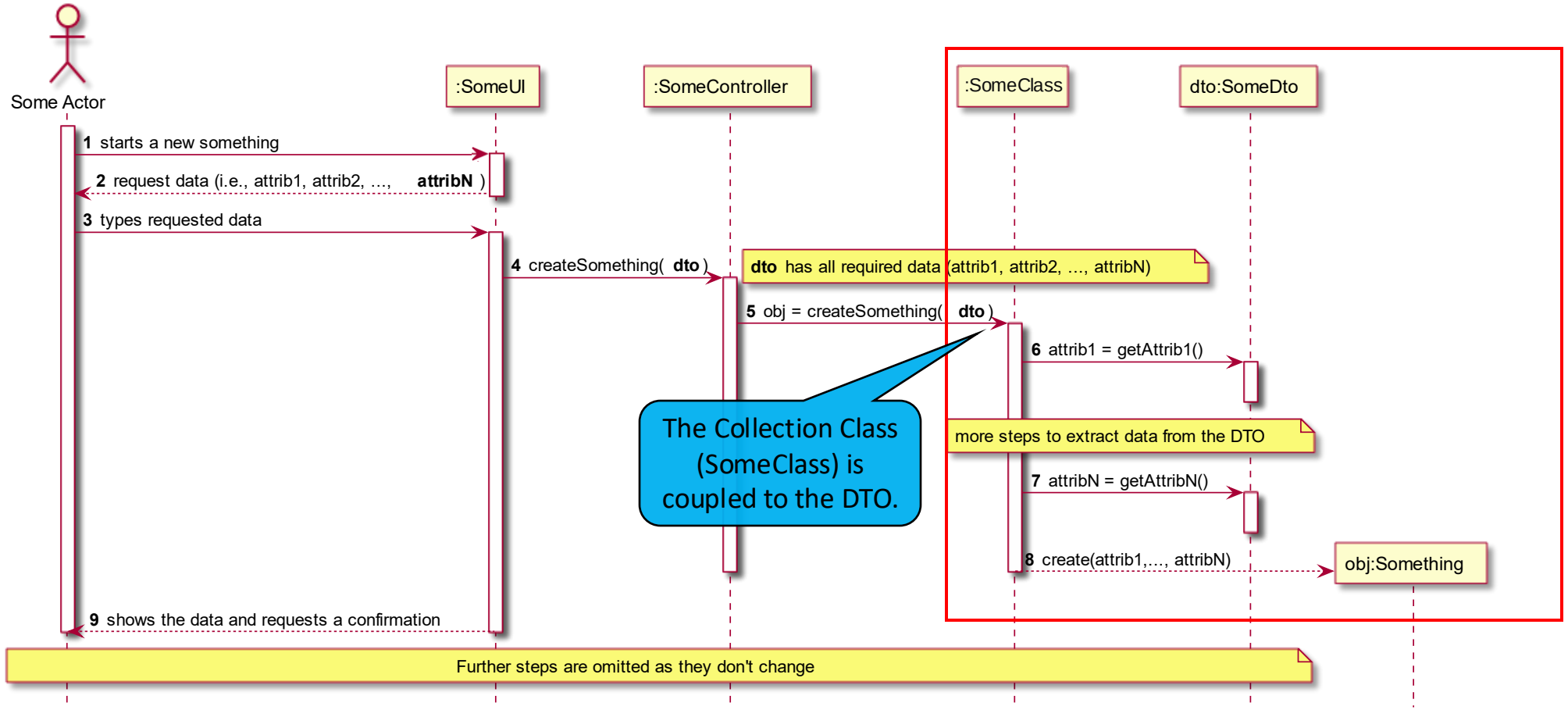
UI → Domain: Sending data to Domain – NO DTO



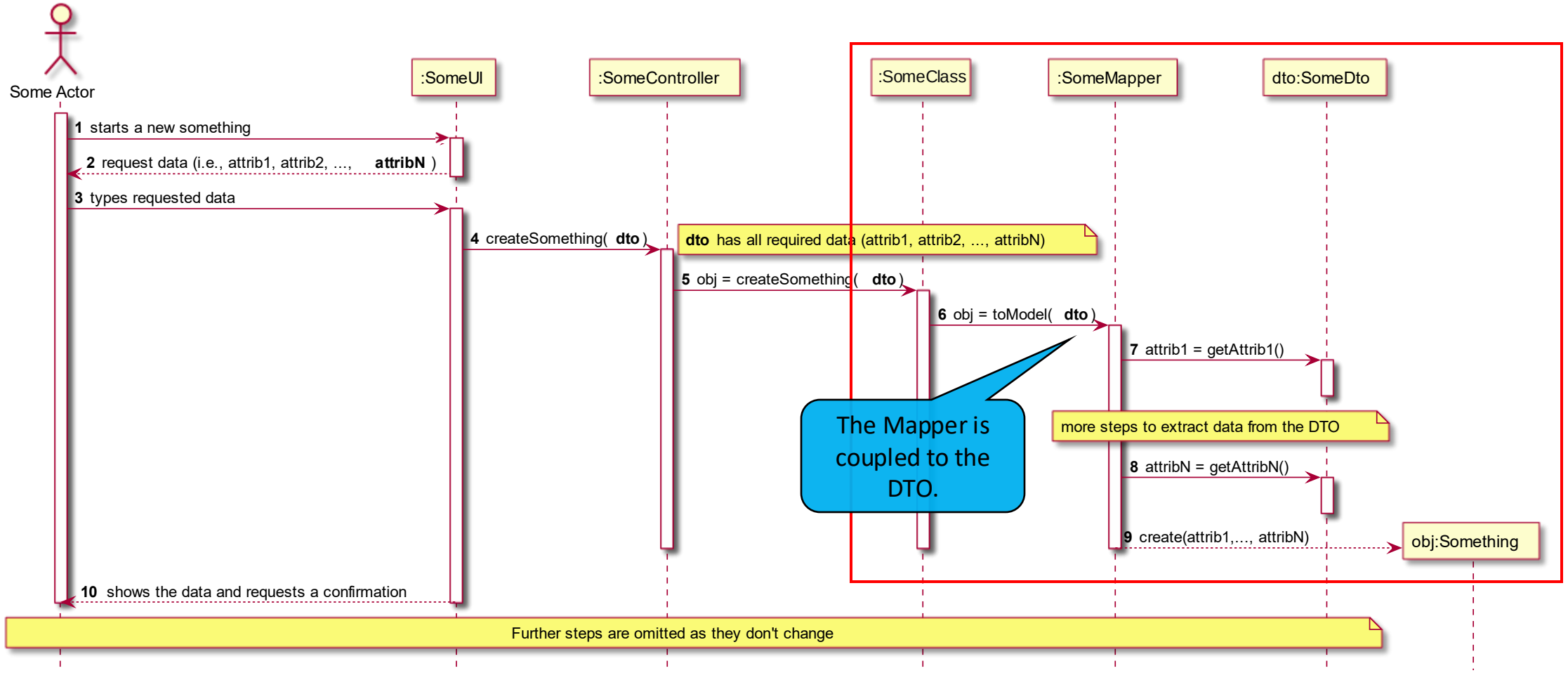
UI → Domain: Sending data to Domain (v1)



UI → Domain: Sending data to Domain (v2)



UI → Domain: Sending data to Domain (v3)



Comparison: v1 vs. v2 vs. v3

- Similarities:
  - The adopted (type of) **DTO is known and expected by the Domain Layer**
  - The Domain Layer **knows how to manipulate** the DTO
  - The Controller compels the UI to send the **expected (type of) DTO** (although not depicted in the previous diagrams)
  - The DTO is used to **stabilize the headers** of involved methods and constructors and to enhance (design and code) readability of such methods/constructors
- The main difference is whether to use (or not) a Mapper
  - On **v1**, the **object being created extracts the data** it needs from the DTO
  - On **v2**, **SomeClass extracts the data** to be able to create the intended object
  - On **v3**, **the data is extracted by a Mapper**, which creates the intended object by delegation from SomeClass (which was previously responsible for creating the object).

Summary

- Use DTO to **reduce the coupling** between layers and increase modularity, reusability and maintainability
- Applying DTO from the Domain Layer to the UI Layer
  - The Domain Layer may not know the exchanged DTO → Only the Controller knows it
  - The DTO provided by the Controller may meet some specific UI needs
  - Immutable domain objects can be used as DTO
- Applying DTO from the UI Layer to the Domain Layer
  - Usually, the Controller compels the UI to send the DTO expected by the Domain Layer
  - In more complex scenarios, the UI may send a DTO (of *TypeX*) to the Controller, which in turn transforms it (using a Mapper) into another DTO (of *TypeY*) which is what the Domain Layer expects
- Use DTO wisely and only in scenarios where it brings some clear advantage – **avoid over-engineering!**

References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- Fowler, Martin; Patterns of Enterprise Application Architecture; Addison Wesley; ISBN-13: 978-0321127426
- https://martinfowler.com/eaaCatalog/dataTransferObject.html