Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Software Engineering

Modeling Dynamic Behavior

The following slides use material from:

Craig Larman; Applying UML and Patterns, 3rd Edition;

Prentice Hall

UML
Interaction Diagrams

Two types of diagrams can be distinguished:

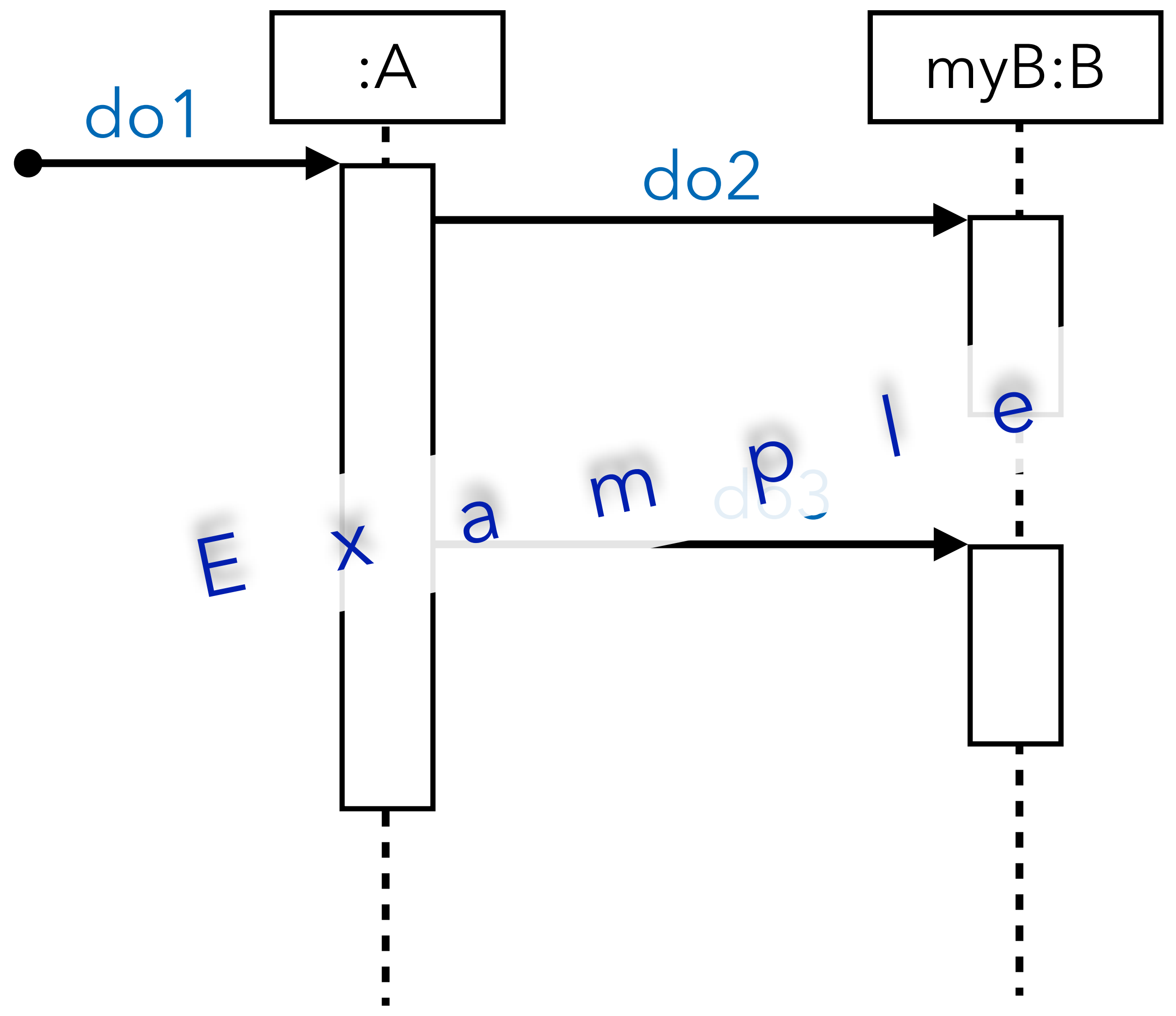
- UML Sequence Diagrams
- UML Communication Diagrams

Interaction diagrams are used to **visualize the interaction via messages between objects**; they are used for *dynamic object modeling*.

Modeling the dynamic behavior is often more rewarding w.r.t. understanding the domain than modeling the static structure.

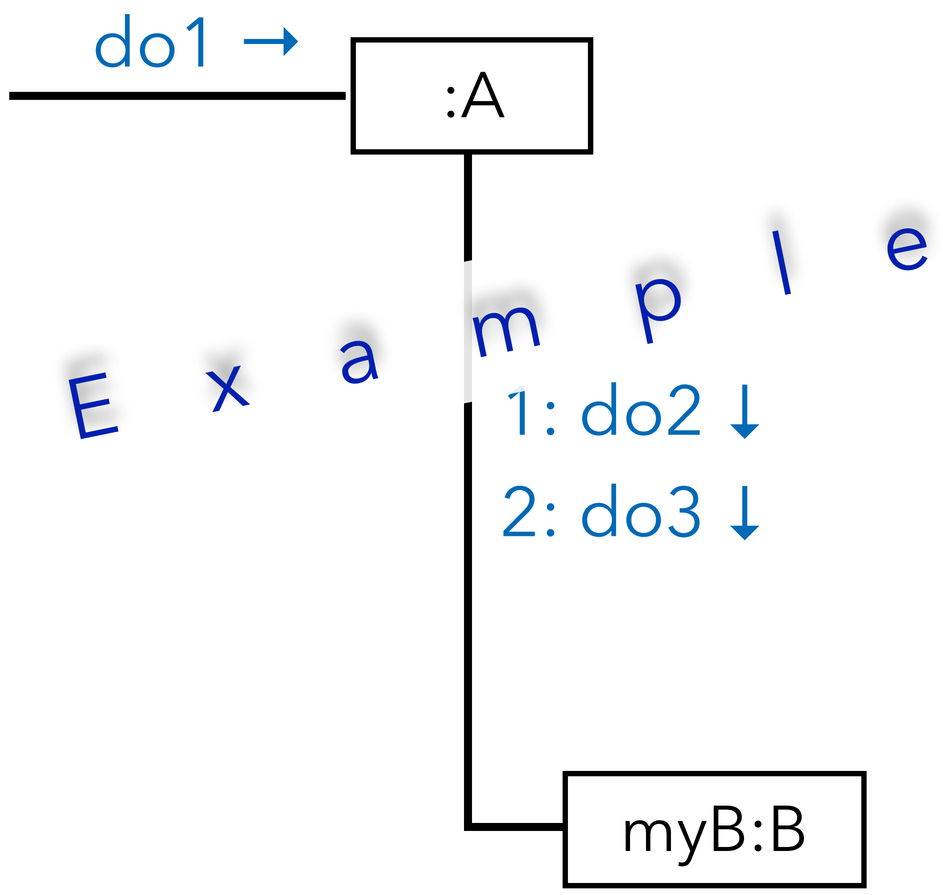
Four types of interaction diagrams are available.

- **Sequence diagrams (which use a fence format.)**
- Communication diagrams (which use a graph or network format)
- Timing diagrams (not discussed)
- Interaction overview diagrams (not further discussed)

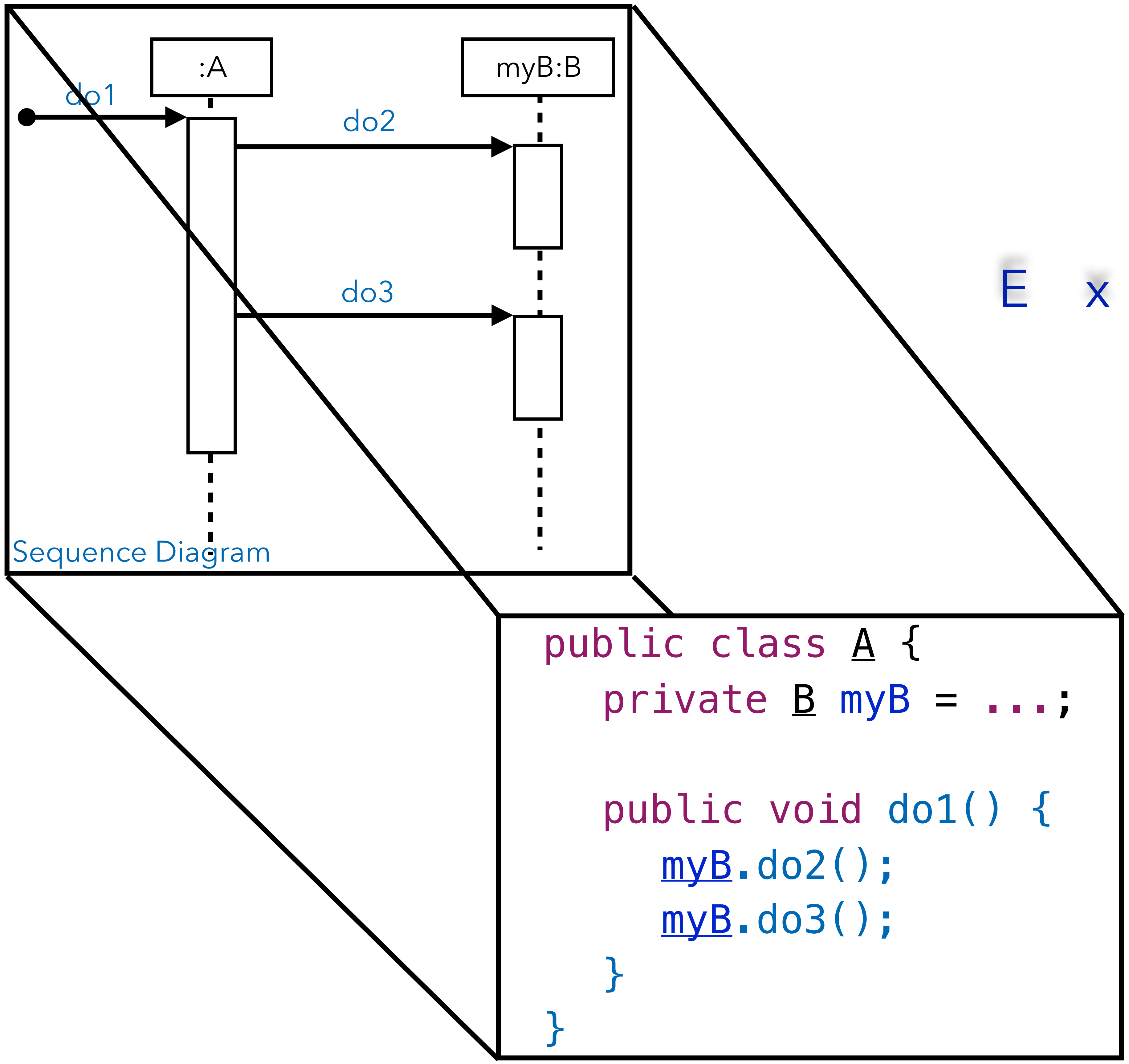


Four types of interaction diagrams are available.

- Sequence diagrams (which use a fence format.)
- **Communication diagrams (which use a graph or network format)**
- Timing diagrams (not further discussed)
- Interaction overview diagrams (not further discussed)



Java Code for Interaction Diagrams

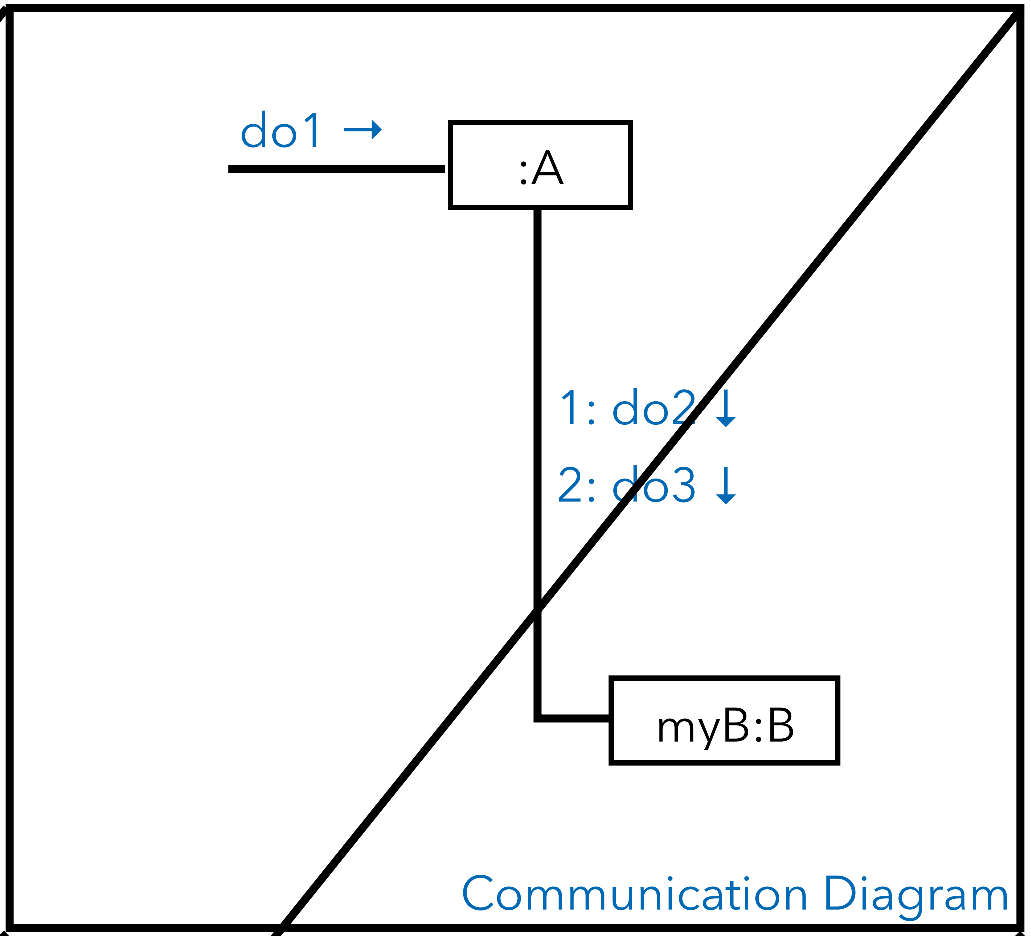


```
public class A {
   private B myB = ...;

   public void do1() {
      myB.do2();
      myB.do3();
   }
}
```

E x a m p l e

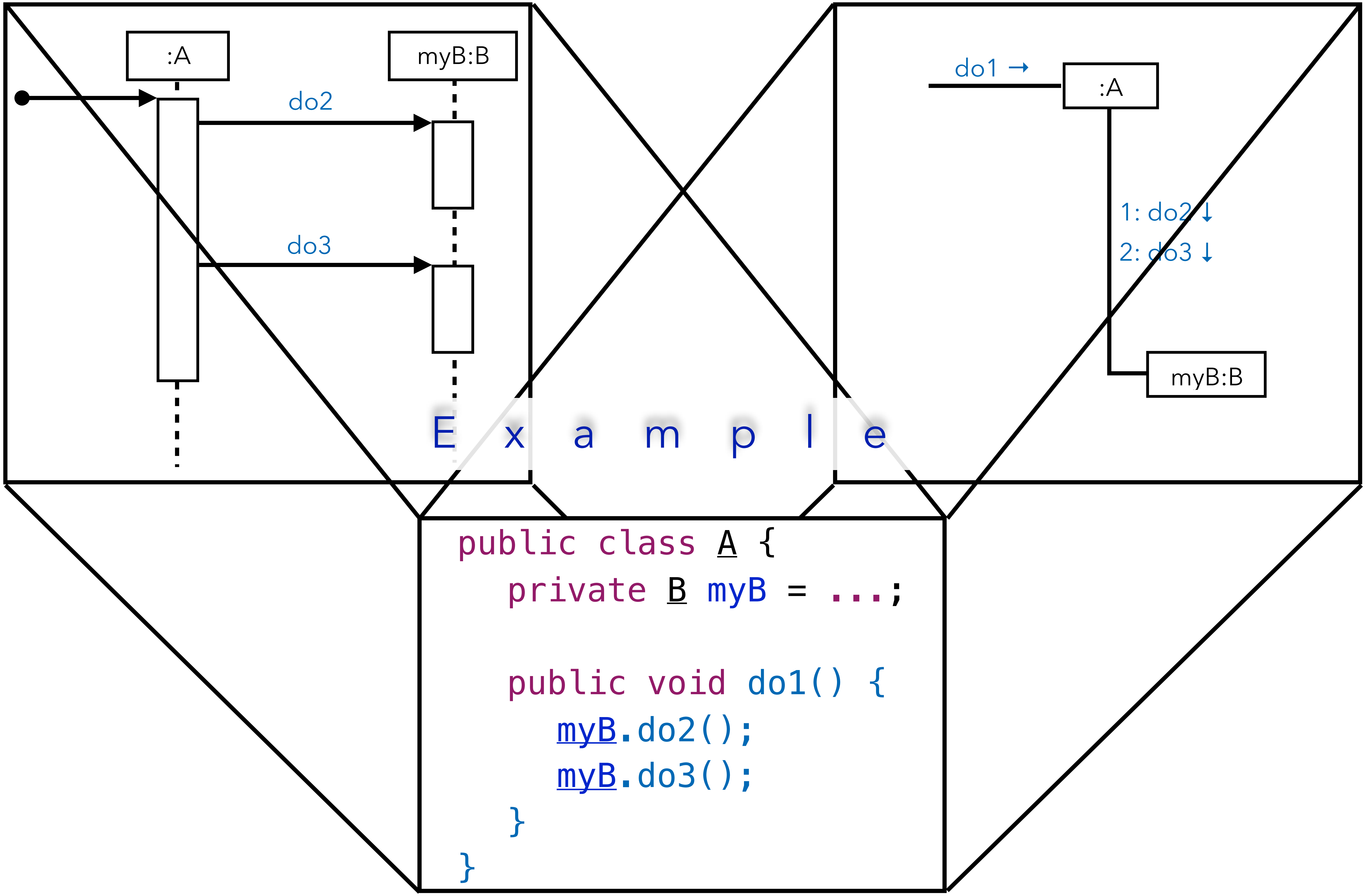
Java Code for Interaction Diagrams

E x a m p l e



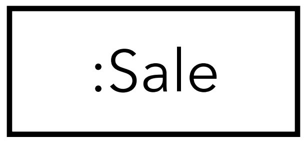
```java
public class A {
   private B myB = ...;

   public void do1() {
      myB.do2();
      myB.do3();
   }
}
```

Java Code for Interaction Diagrams



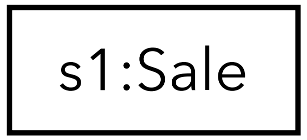
```java
public class A {
   private B myB = ...;

   public void do1() {
      myB.do2();
      myB.do3();
   }
}
```

Common Notations for UML Interaction Diagrams



Lifeline box representing an unnamed instance of class Sale.

Common Notations for UML Interaction Diagrams

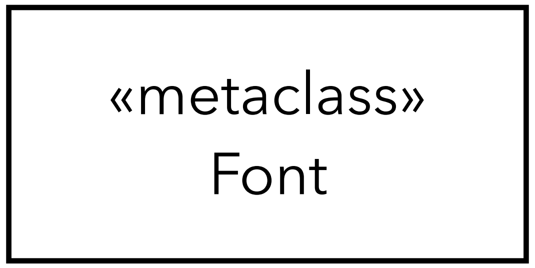


Java Code:

```
Sale s1 = …;
```

Lifeline box representing a named instance (s1) of Sale.

Common Notations for UML Interaction Diagrams

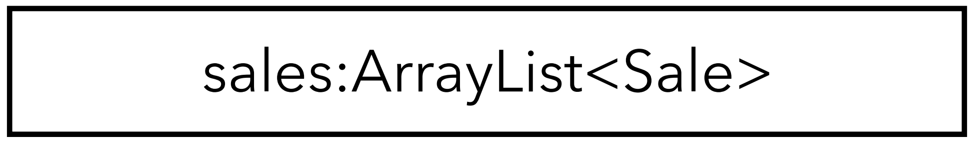


Java Code:

```
Class<Font> fontClass = Font.class;
```

Lifeline box representing the class Font, or more precisely, that Font is an instance of class Class - an instance of a metaclass.

Common Notations for UML Interaction Diagrams

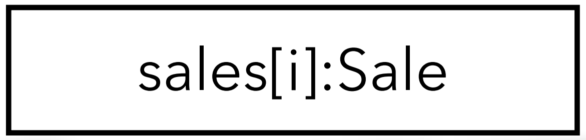


Java Code:

```
ArrayList<Sale> sales = …;
```

Lifeline box representing an instance of an ArrayList class, parameterized to hold Sale objects.

Common Notations for UML Interaction Diagrams

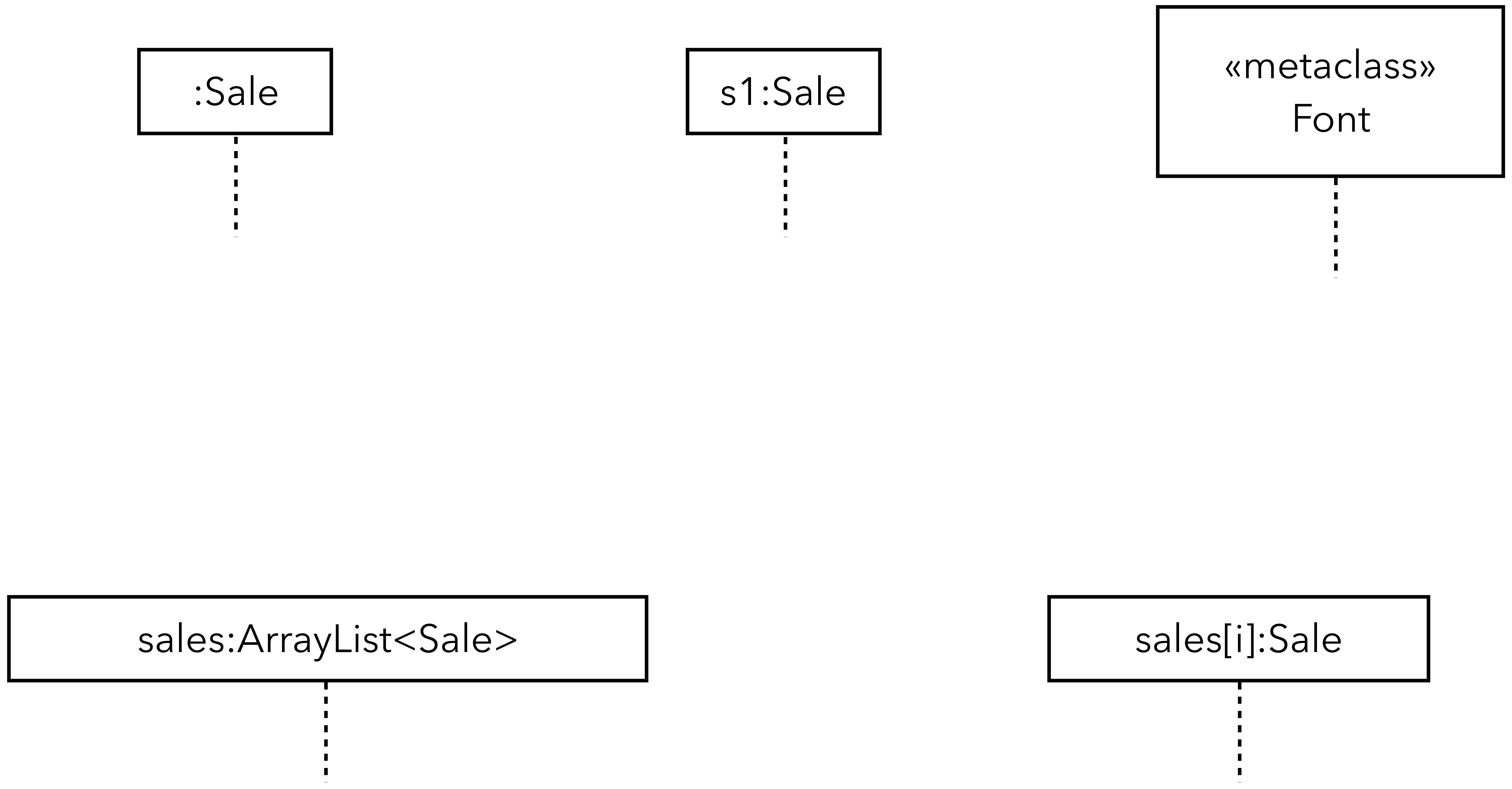


Java Code:

```
ArrayList<Sale> sales = …;
Sale sale = sales.get(i);
```

Lifeline box representing one instance of class Sale, selected from the sales ArrayList<Sale> collection.

Common Notations for UML Interaction Diagrams



O v e r v i e w

Common Notations for UML Interaction Diagrams - Format for Interaction Messages

- "Commonly" Used Grammar:
  return = message(parameter:parameterType):returnType
- Parentheses are usually excluded if there are no parameters.
- Type information may be excluded if unimportant.

Examples

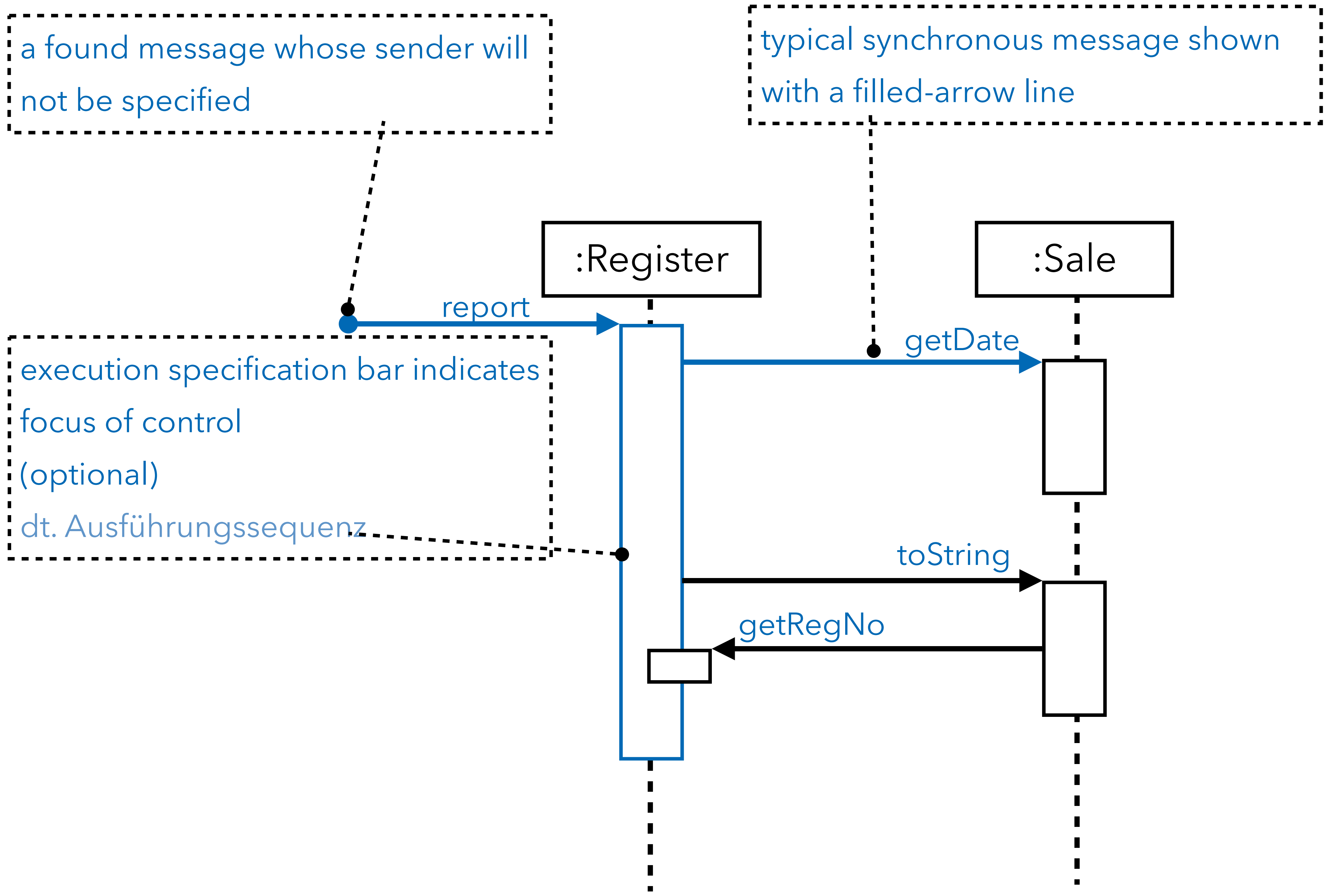
```
initialize(code)
initialize
d = getProductDescription (id)
d = getProductDescription (id : ItemId)
d = getProductDescription (id : ItemId) : ProductDescription
```

The same syntax is used by, e.g., the Scala programming language.

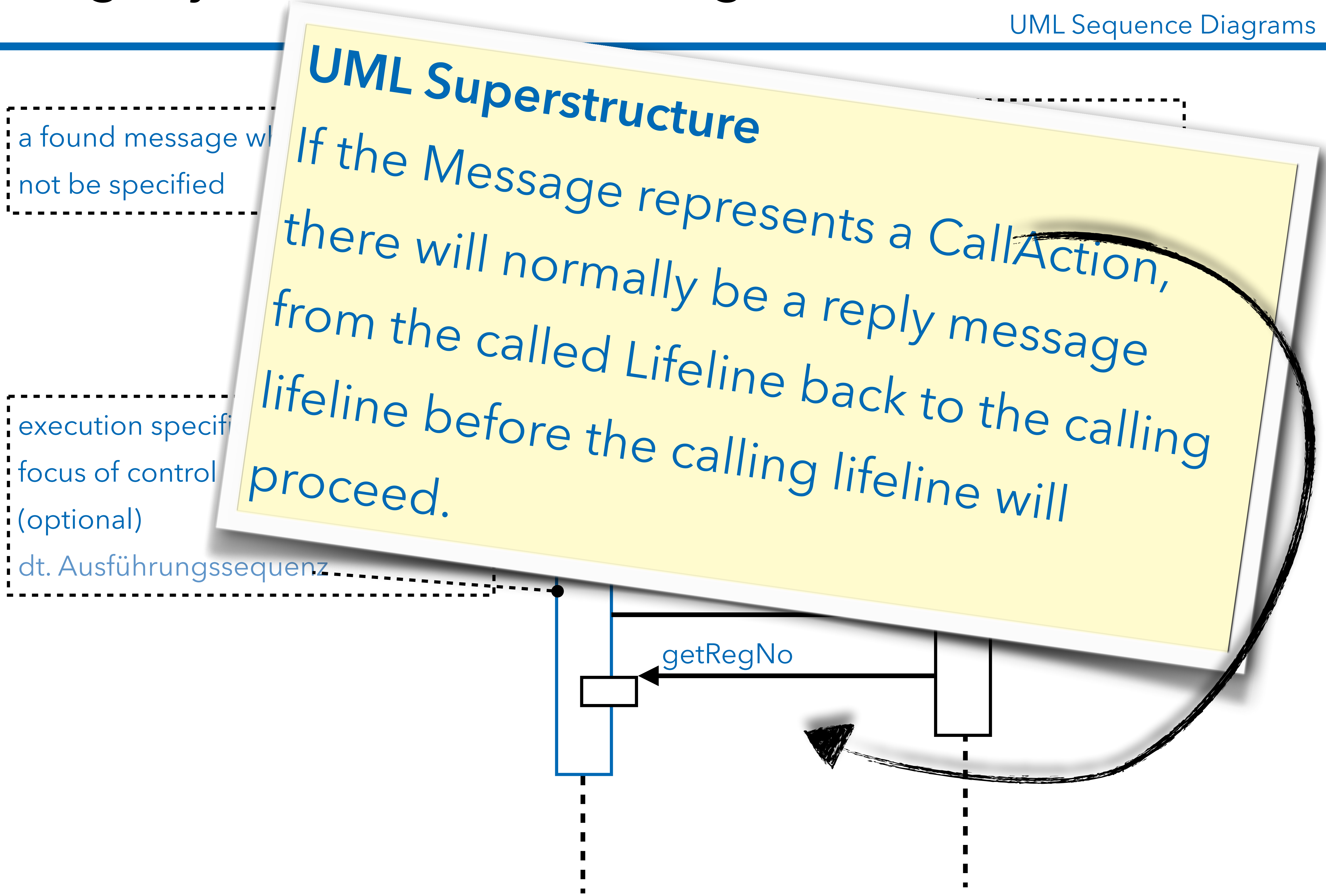
UML
Sequence Diagrams

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Modeling (Synchronous) Messages


a found message whose sender will not be specified
typical synchronous message shown with a filled-arrow line
:Register
:Sale
report
getDate
execution specification bar indicates focus of control (optional)
dt. Ausführungssequenz
toString
getRegNo

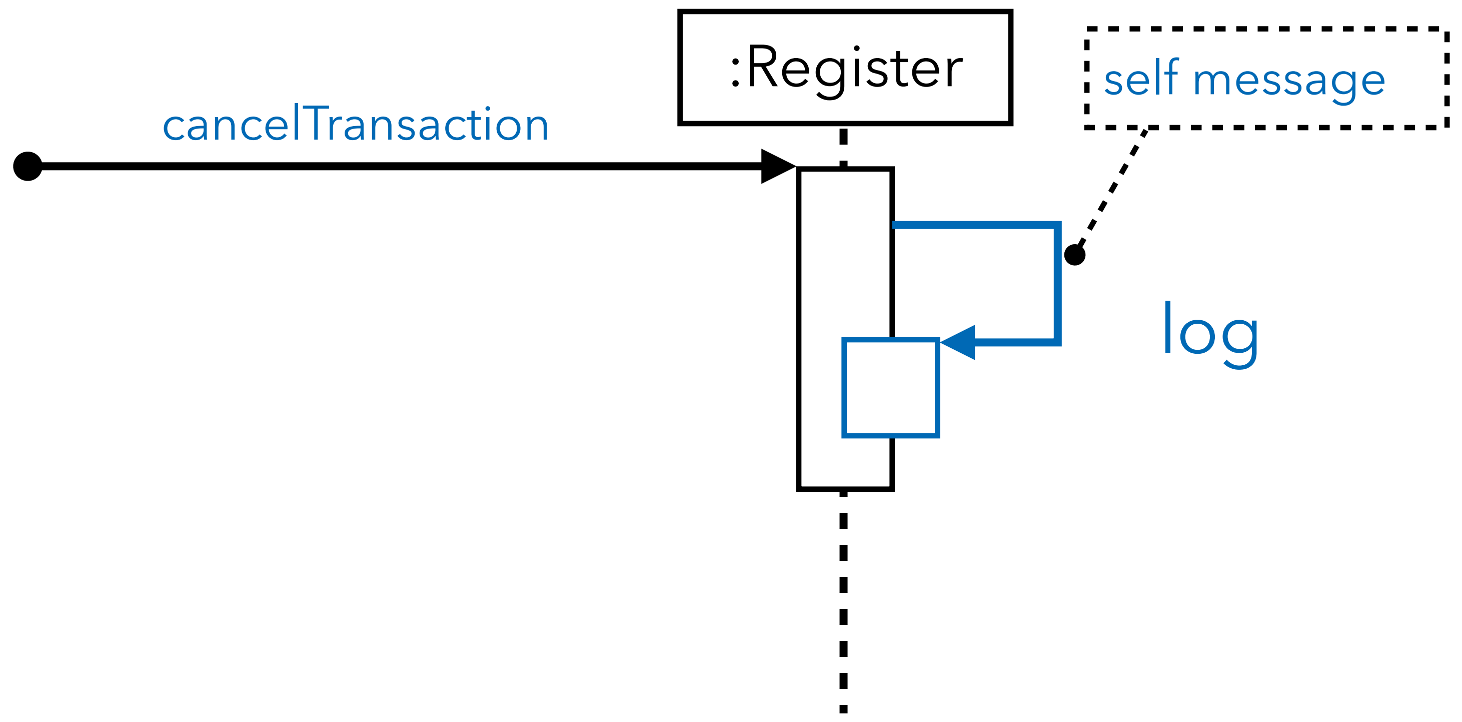
Modeling (Synchronous) Messages



UML Superstructure

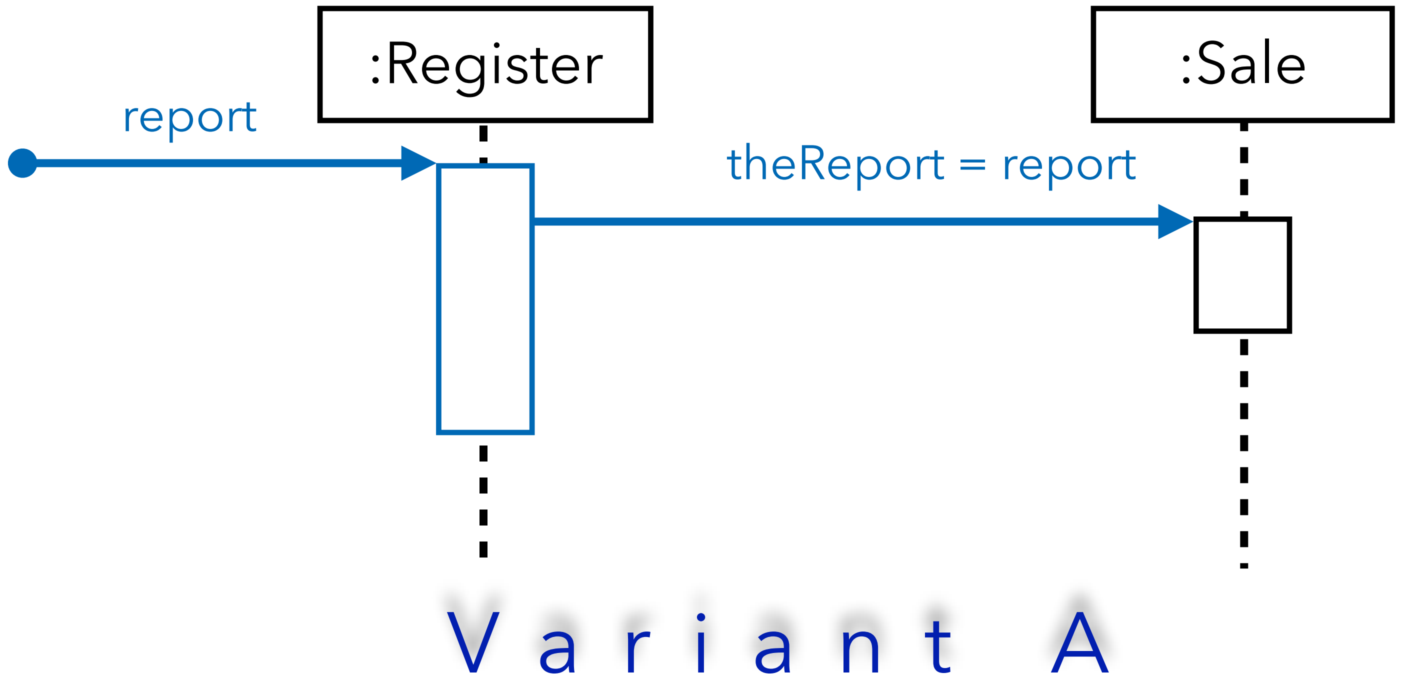
If the Message represents a CallAction, there will normally be a reply message from the called Lifeline back to the calling lifeline before the calling lifeline will proceed.

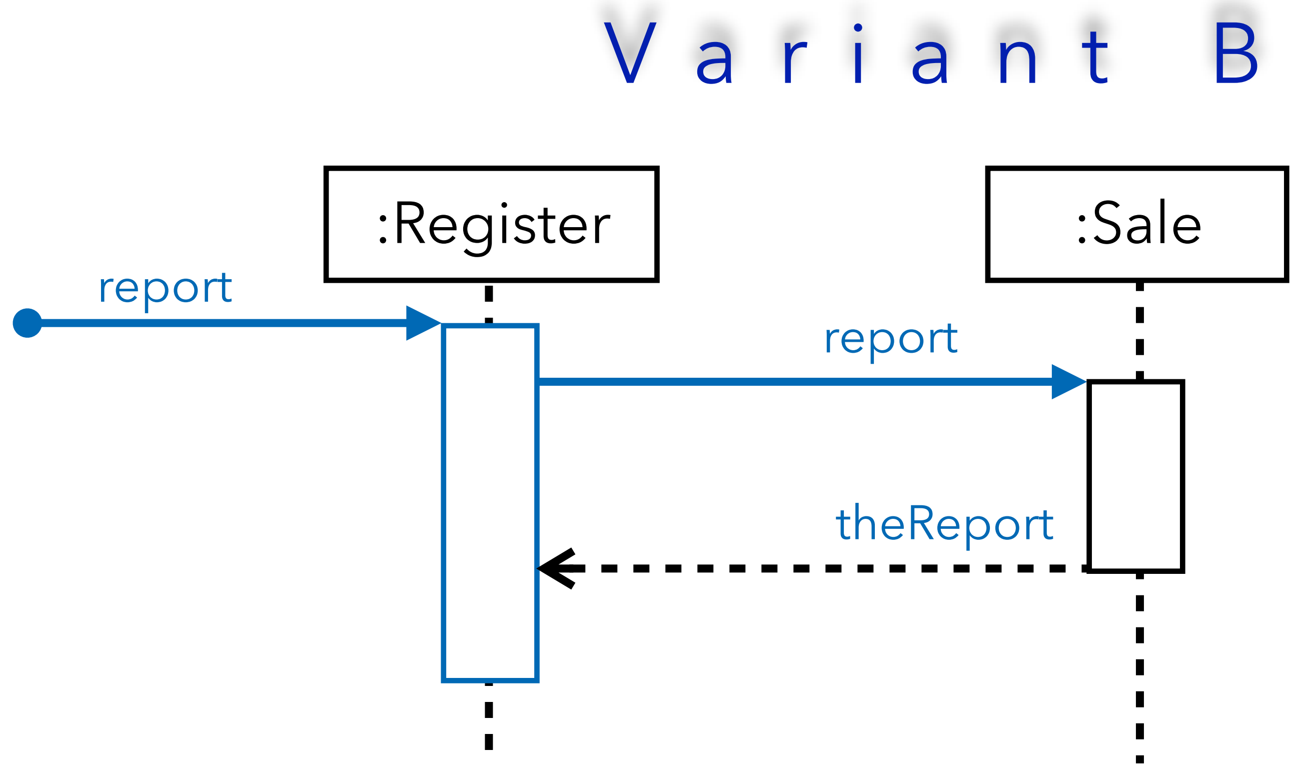
Self messages can be modeled using nested execution specification bars.



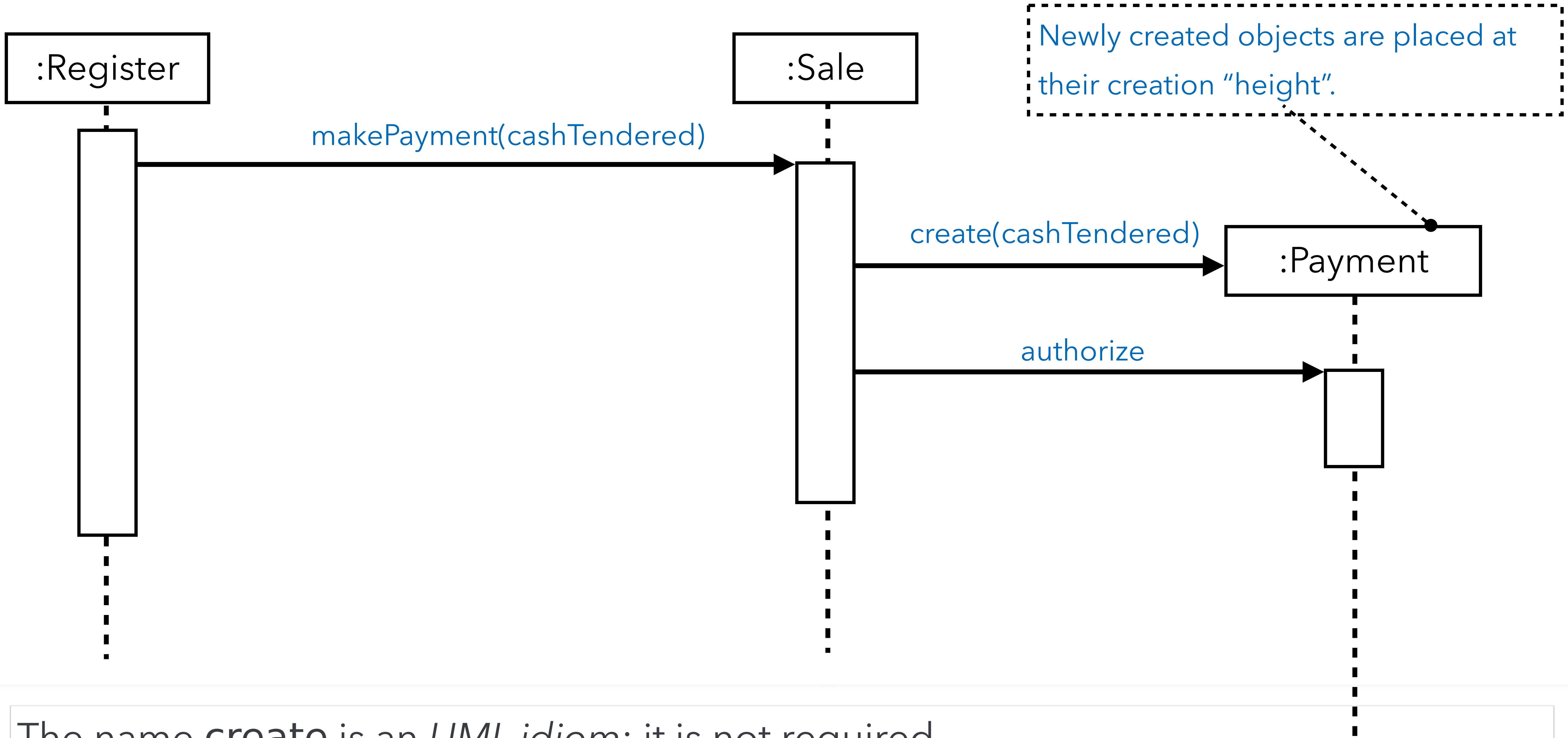
To show the return value of a message you can either use the message syntax (A) or use a message line at the end of an execution specification bar (B).

execution specification bar =dt. Ausführungssequenz



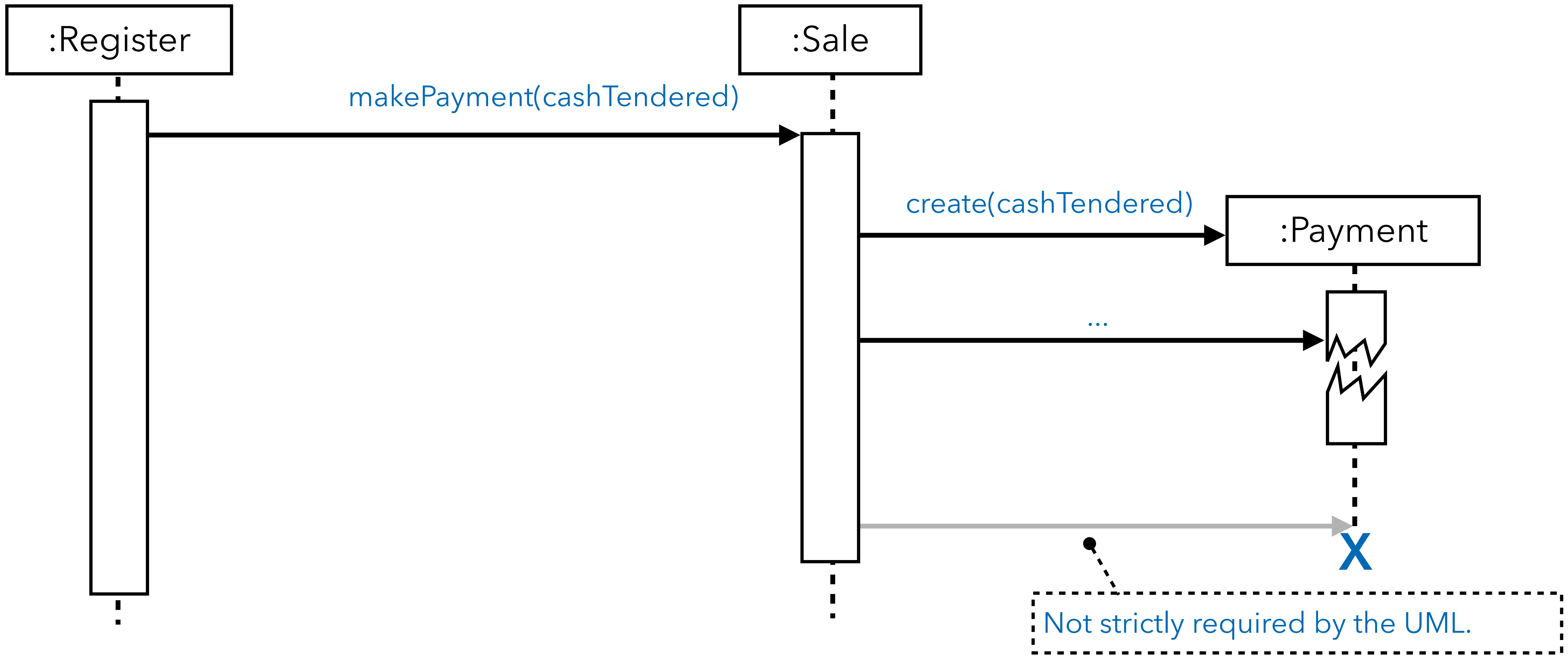


Object Instance Creation


:Register
:Sale
makePayment(cashTendered)
create(cashTendered)
:Payment
authorize
Newly created objects are placed at their creation "height".

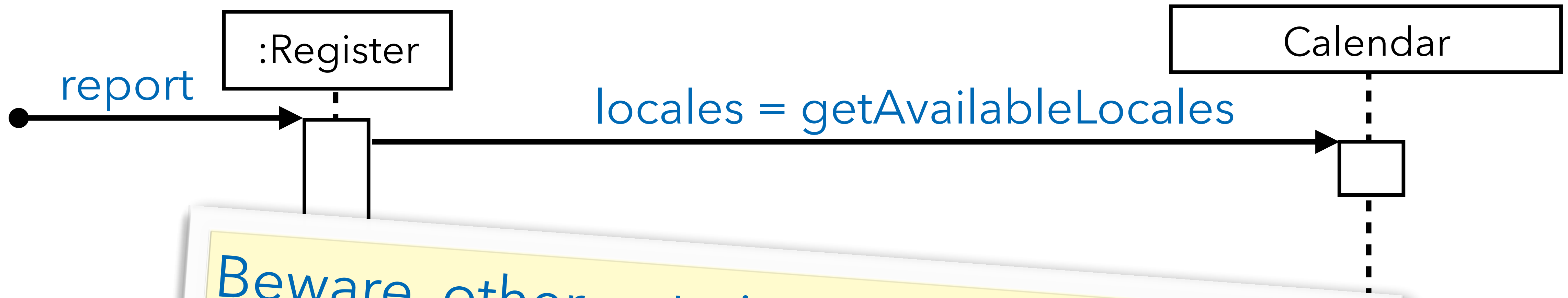
The name **create** is an *UML idiom*; it is not required.

Object Instance Destruction



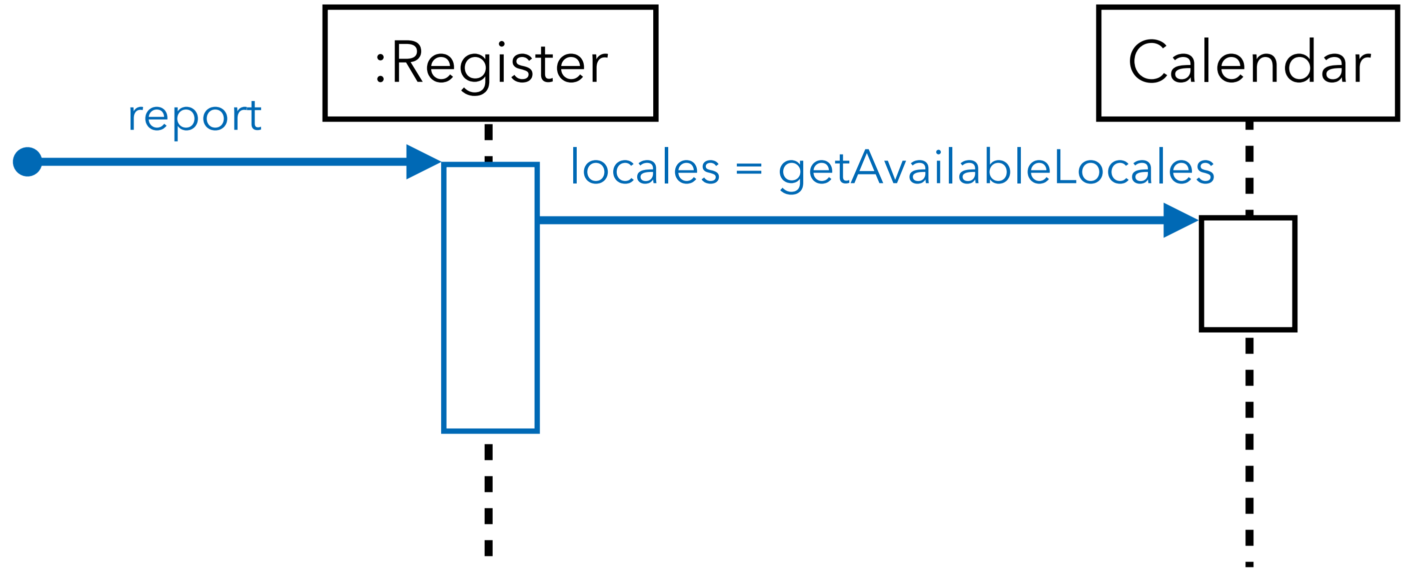
The object destruction notation is also used to mark objects that are no longer usable.

Invoking Static Methods (Class Methods)



Beware, other notations are also used (e.g. underlined method names).

Invoking Static Methods (Class Methods)

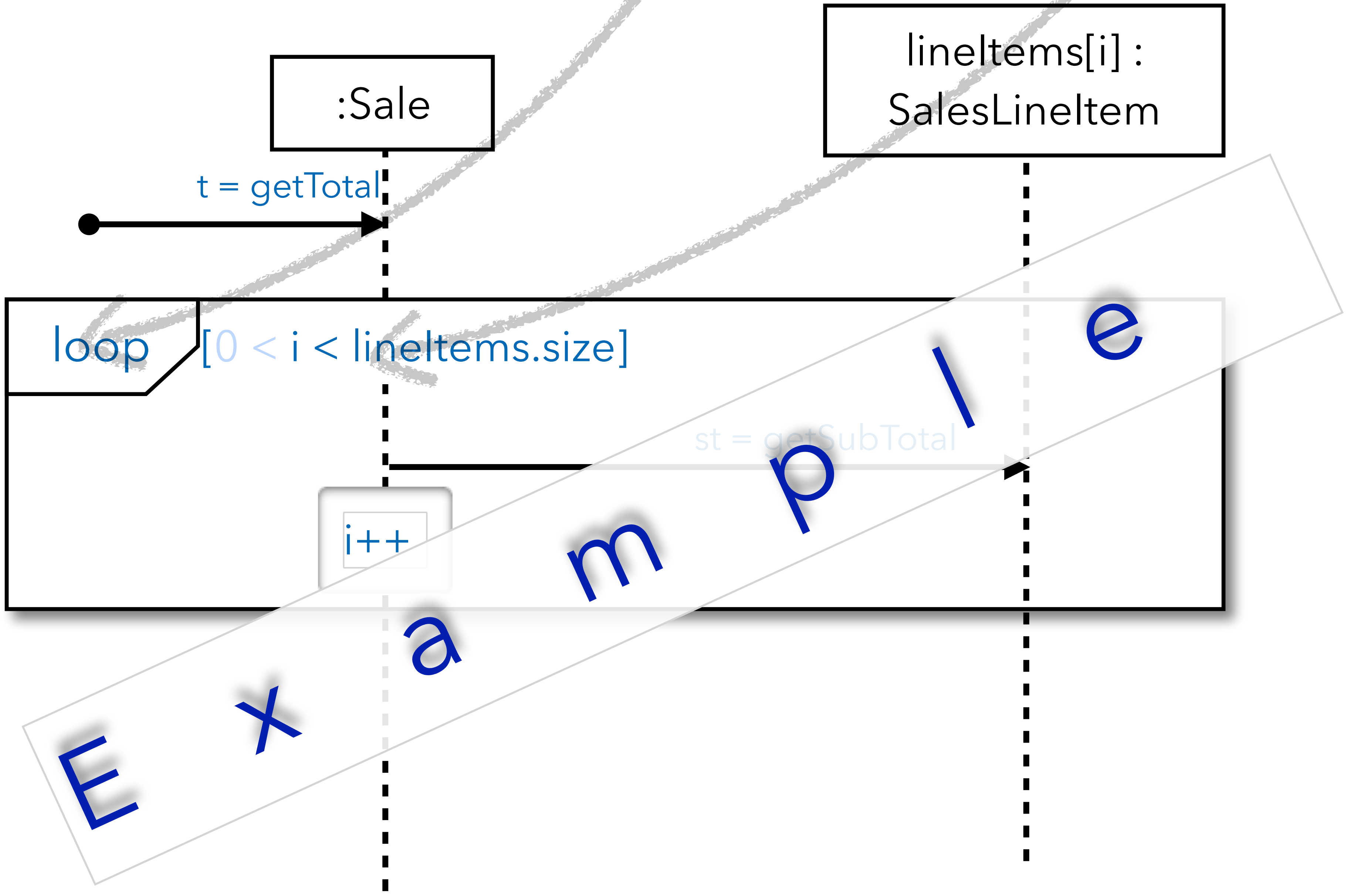


```
public class Register {
   public void report() {
      Locale[] locales = Calendar.getAvailableLocales();
   }
}
```

Corresponding Java Code

Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs. Frames have an operator and a guard.

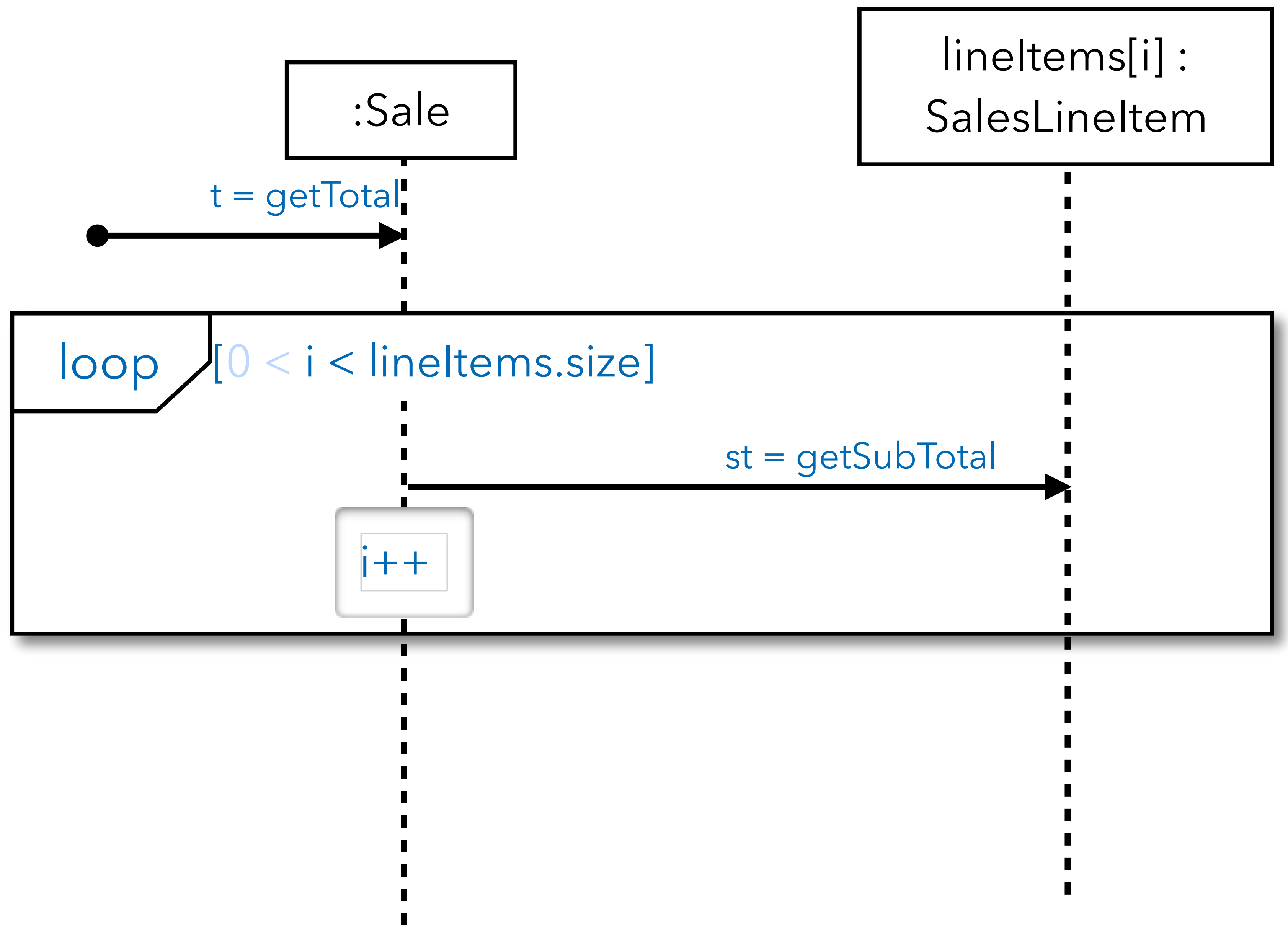
Diagram Frame ~dt. Fragment



How to model the iteration over a collection?

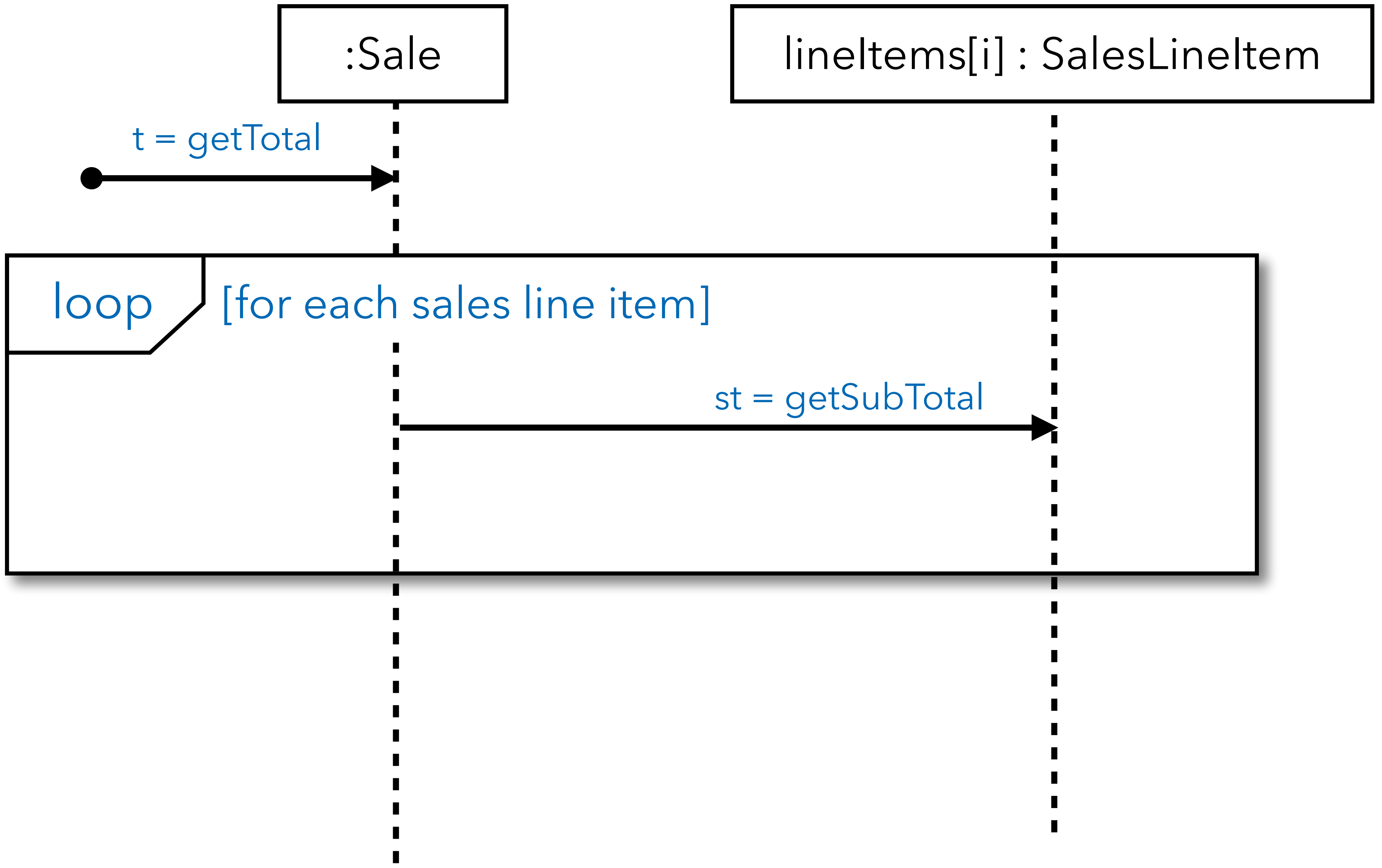
```
Modeling task: Calculate the total of a sale by summing up the
sub totals for each sales line item.
```

Use a **UML loop frame** to iterate over a collection.



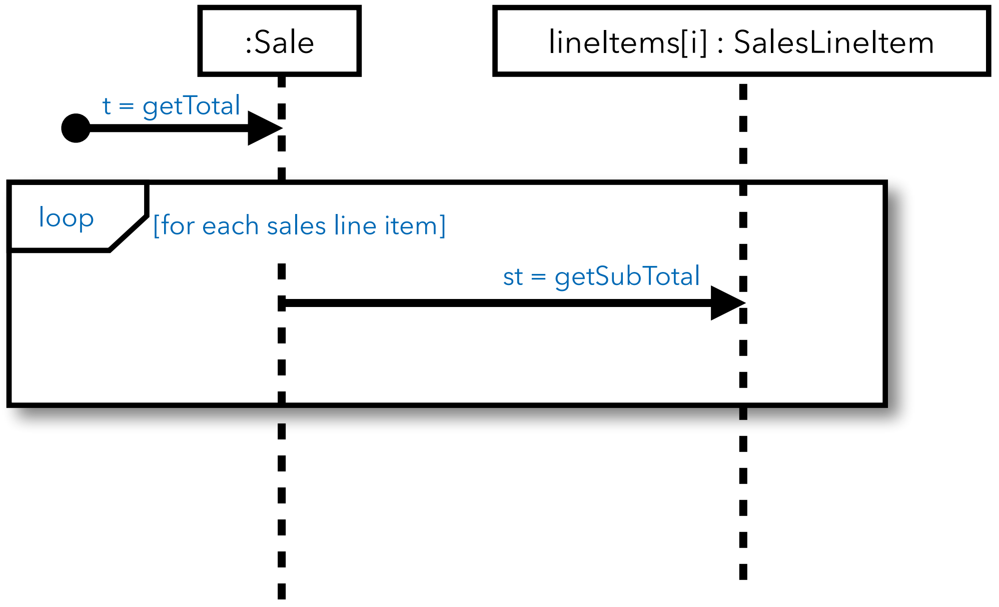
Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.

Use a **UML loop frame** to iterate over a collection.



```
Modeling task: Calculate the total of a sale by summing up the
sub totals for each sales line item.
```

Java code corresponding to a **UML loop frame**.



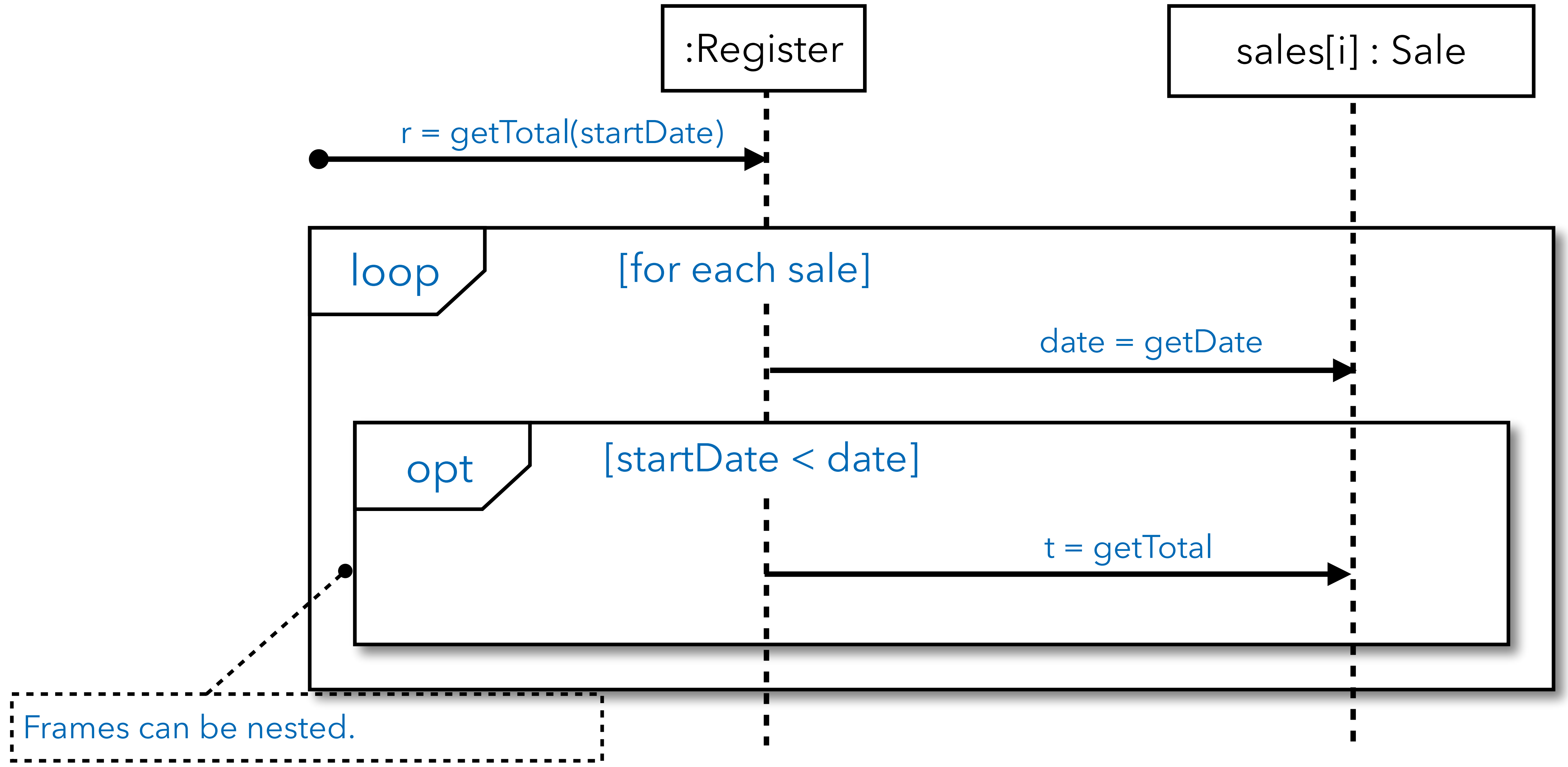
```
public class Sale {

   private List<SalesLineItem> lineItems
   = new ArrayList<SalesLineItem>();

   public Money getTotal() {
      Money t = new Money();
      Money st = null;
      for (SalesLineItem lineItem : lineItems) {
         st = lineItem.getSubtotal();
         t.add(st);
      }
      return t;
   }

}
```

```
Modeling task: Calculate the total of a sale by summing up the
sub totals for each sales line item.
```

How to model the sending of a message only if a guard condition matches?

```
Modeling task: Get the sum of all sales that happened today
after 18:00 o’clock.
```

Use a **UML opt frame** to model the sending of a message if the guard condition matches.

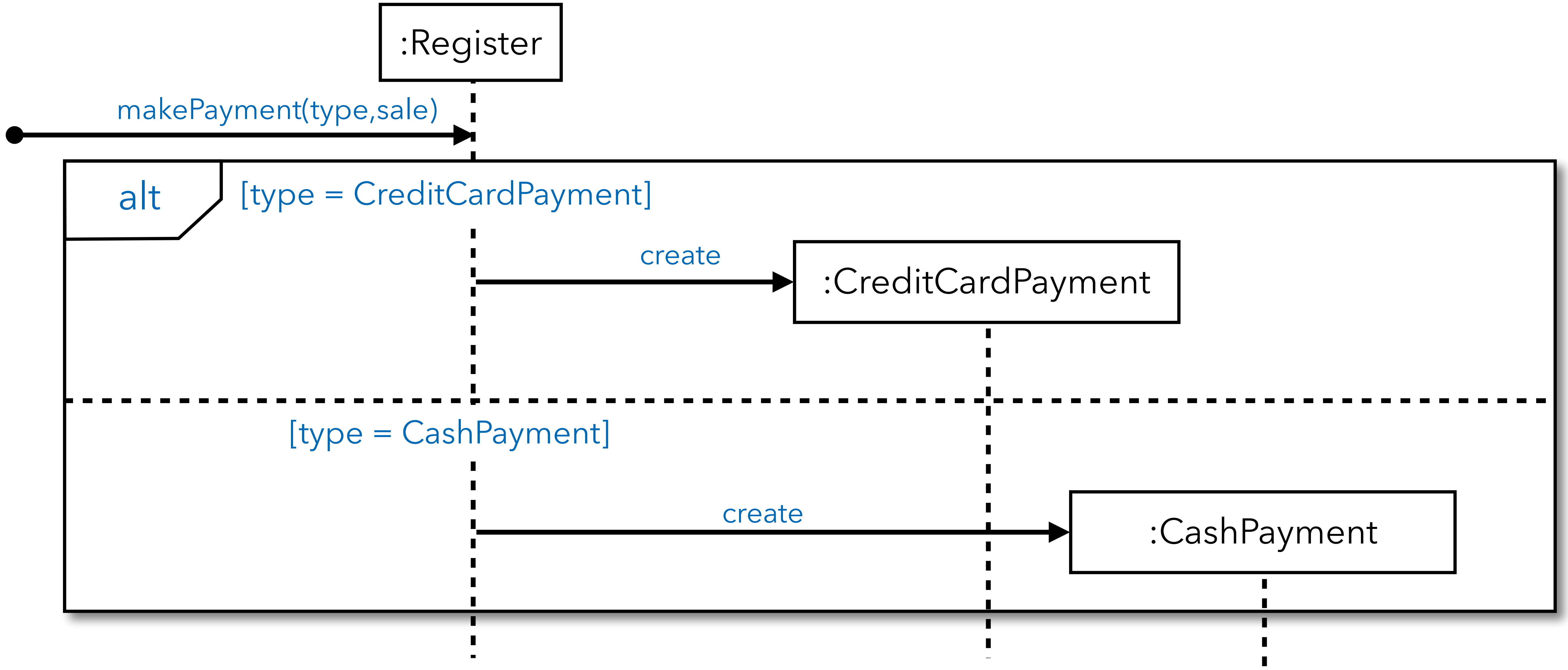


```
Modeling task: Get the sum of all sales that happened today
after 18:00 o'clock.
```

How to model mutually exclusive alternatives?

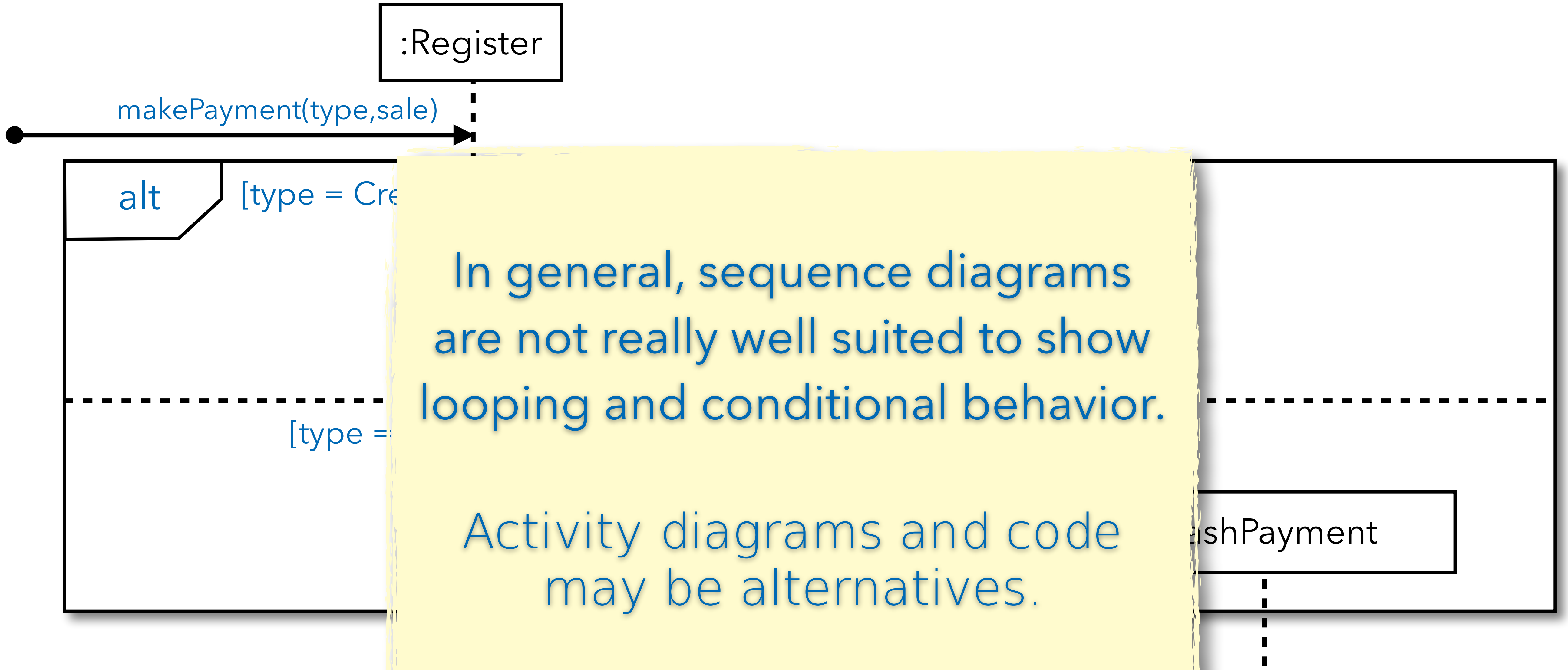
```
Modeling task: A register should be able to handle credit card
payments and cash payments.
```

Use the **UML alt frame** to model between 2 and n mutually exclusive alternatives.



```
Modeling task: A register should be able to handle credit card
payments and cash payments.
```

Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs.



In general, sequence diagrams are not really well suited to show looping and conditional behavior.

Activity diagrams and code may be alternatives.

```
Modeling task: A register should be able to handle credit card
payments and cash payments.
```

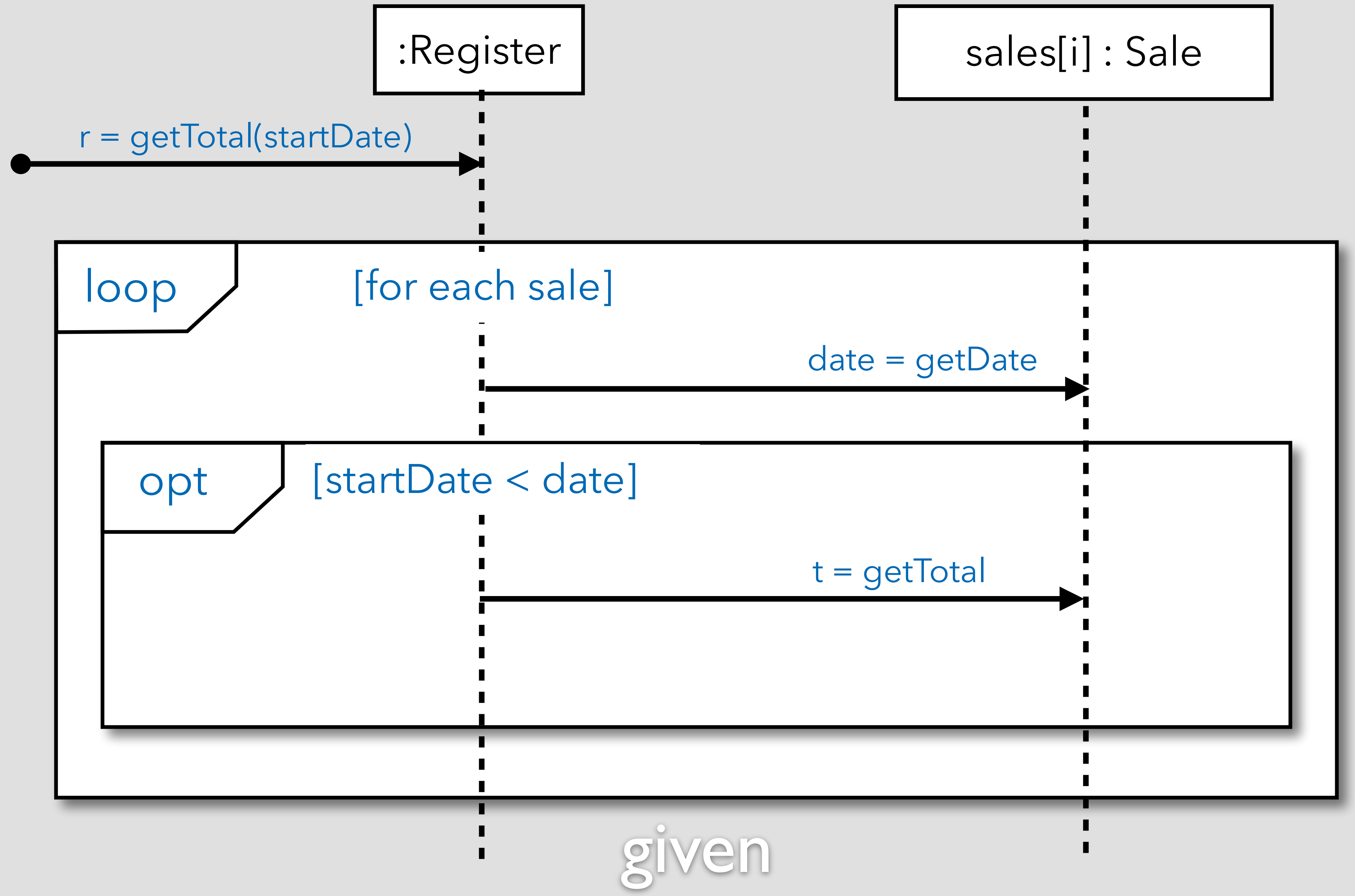
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

- References are used to simplify a diagram and factor out a portion into another diagram or to enable reuse.

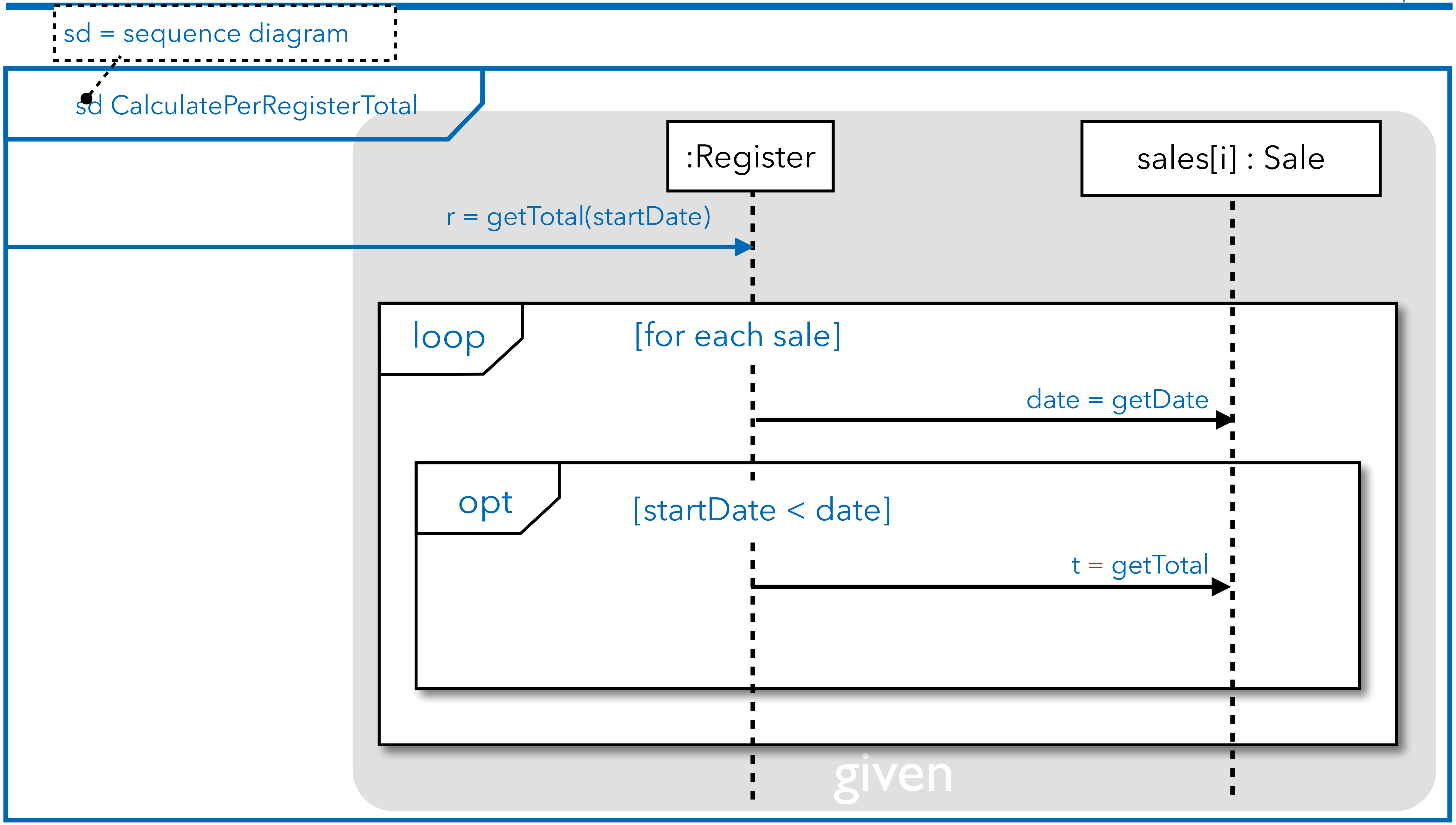
```
Modeling task: We want to calculate the store's overall total.
```

interaction occurrence ~dt. Interaktionsauftreten

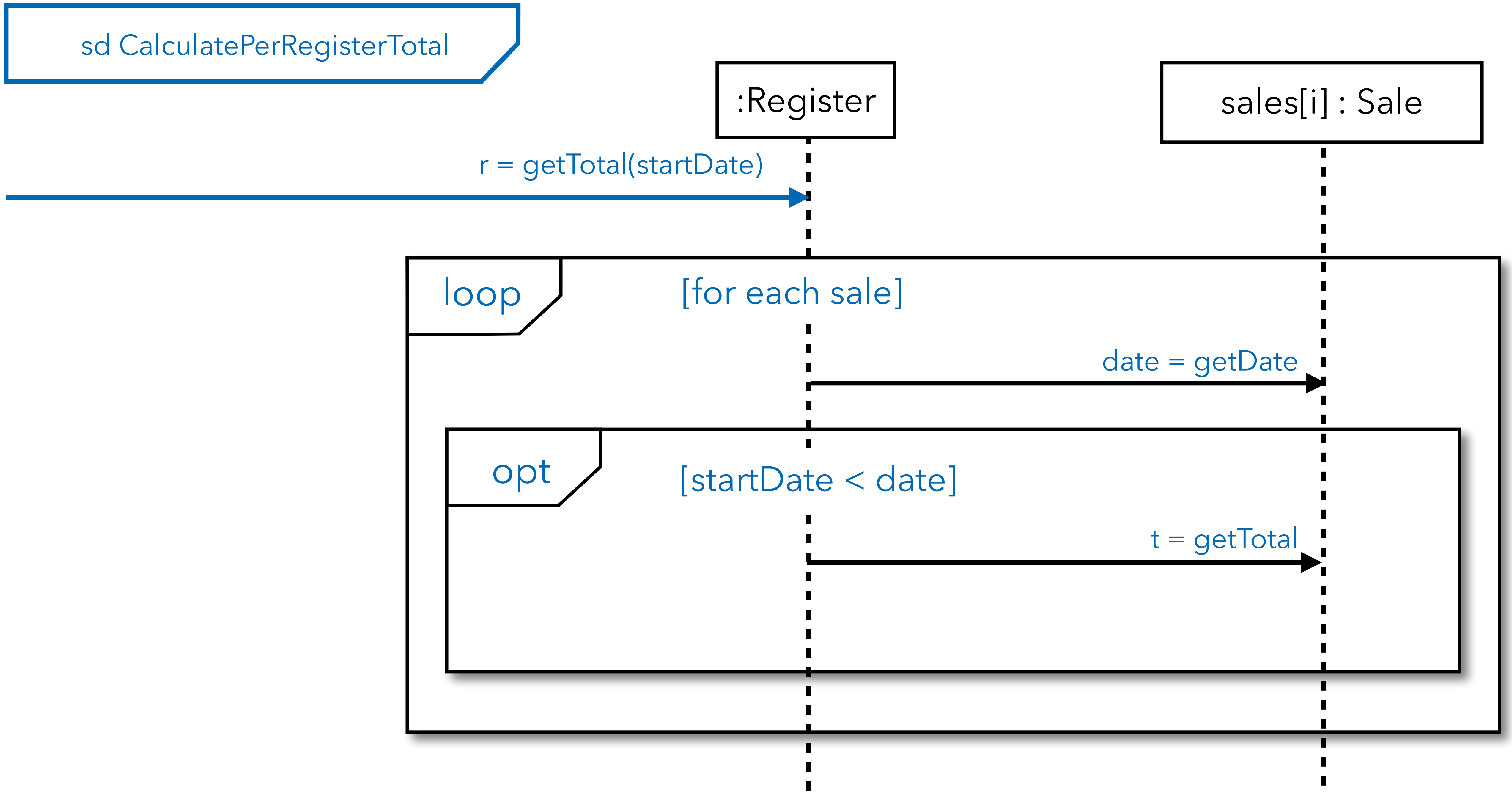
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.



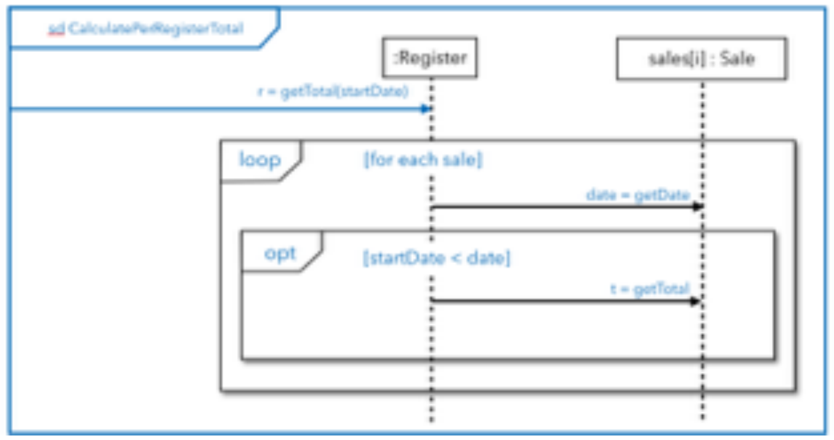
An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

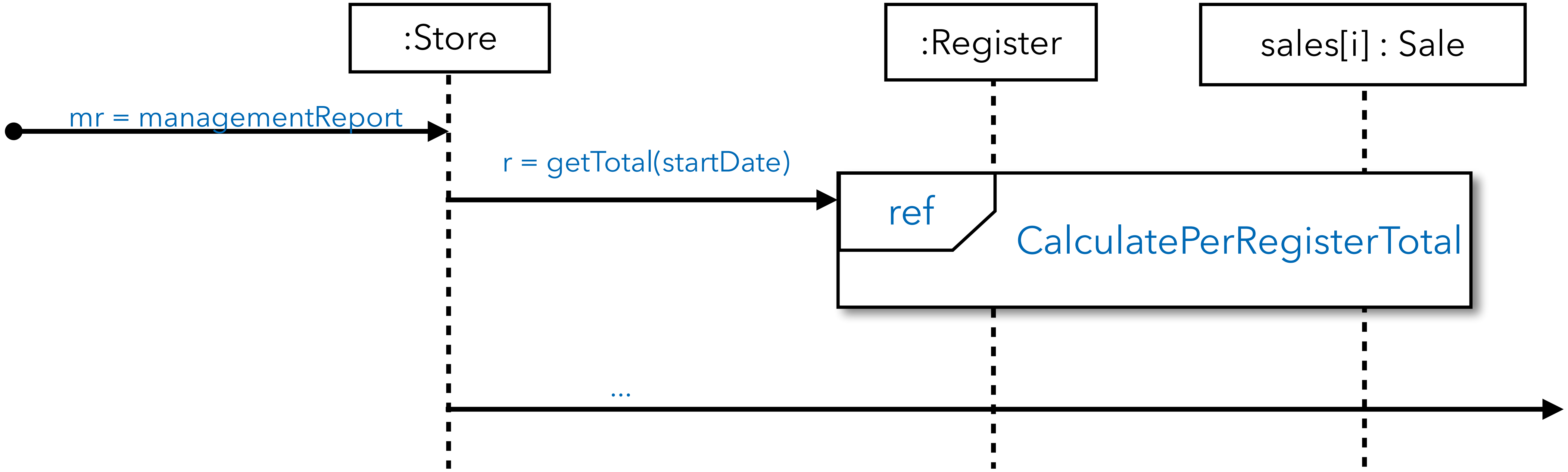


An interaction occurrence (interaction use) is a reference to an interaction within another interaction.



An interaction occurrence (interaction use) is a reference to an interaction within another interaction.

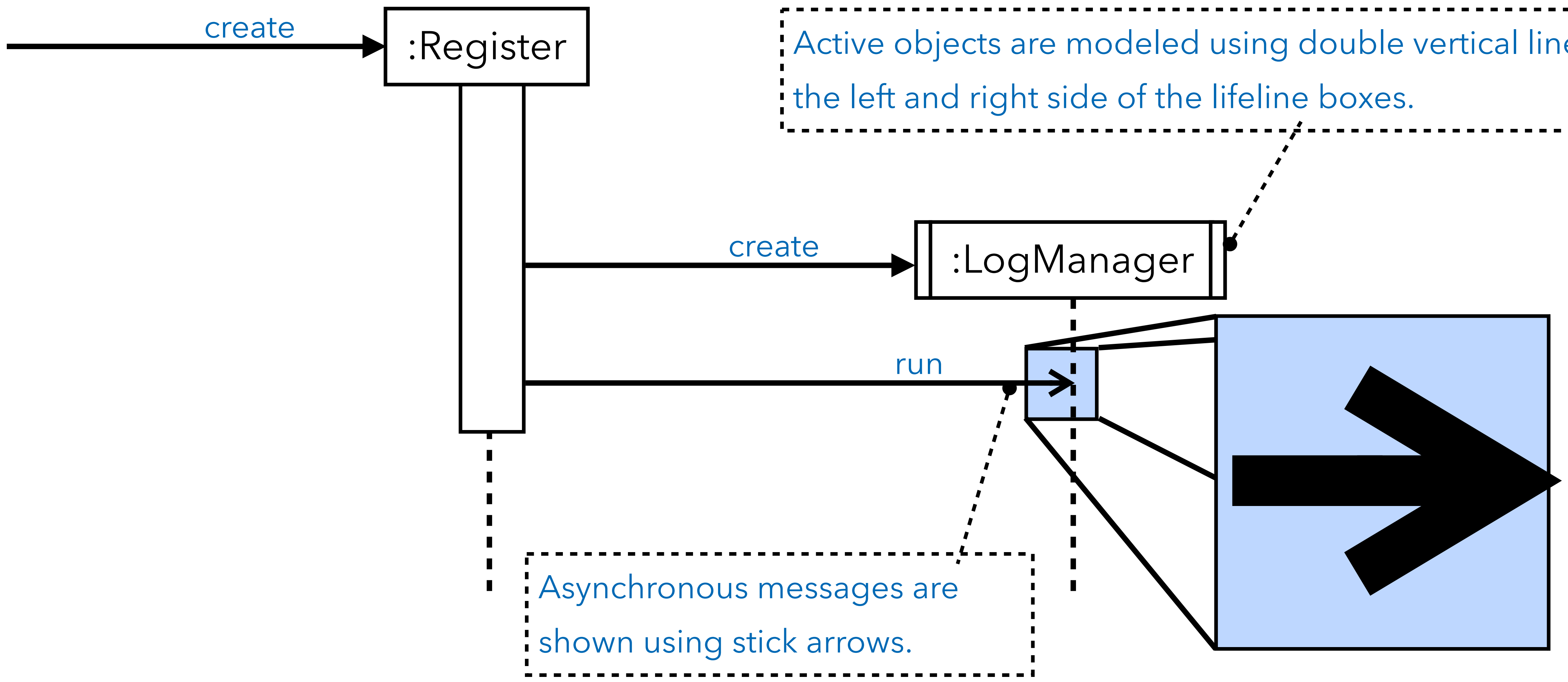




How to model the sending of asynchronous messages?
How to model objects that have their own thread of execution?

```
Modeling task: The log information should automatically be
collected and processed in the background.
```

Asynchronous messages are messages that don't block. An **active object** is an object where each instance runs on and controls its own thread of execution.


create
:Register
Active objects are modeled using double vertical lines on the left and right side of the lifeline boxes.
create
:LogManager
run
Asynchronous messages are shown using stick arrows.

```
Modeling task: The log information should automatically be
collected and processed in the background.
```

UML Communication Diagrams

TECHNISCHE UNIVERSITÄT DARMSTADT

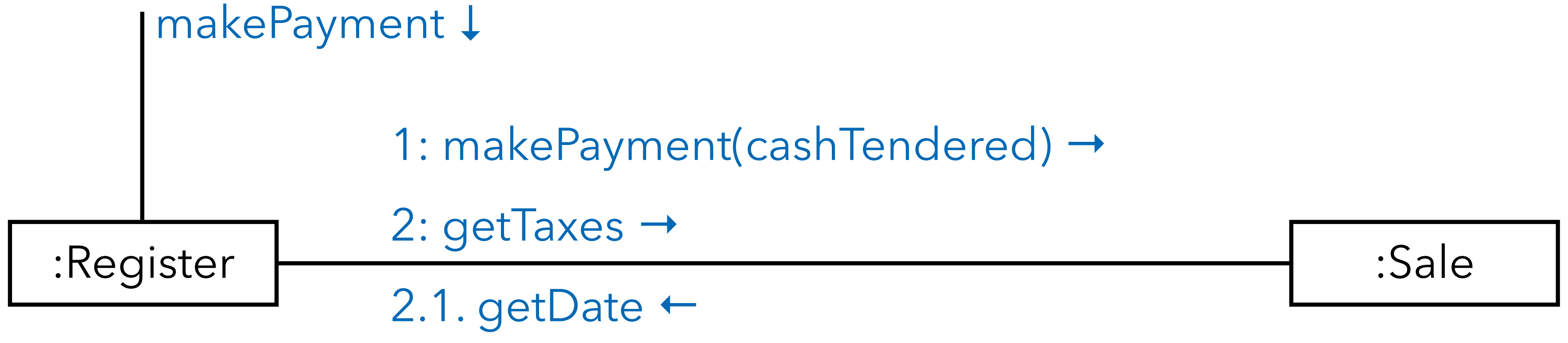
Links and Messages in Communication Diagrams

- A link is a connection path between two objects (it is an instance of an association)

  A link indicates that some form of navigation and visibility between the objects is possible.

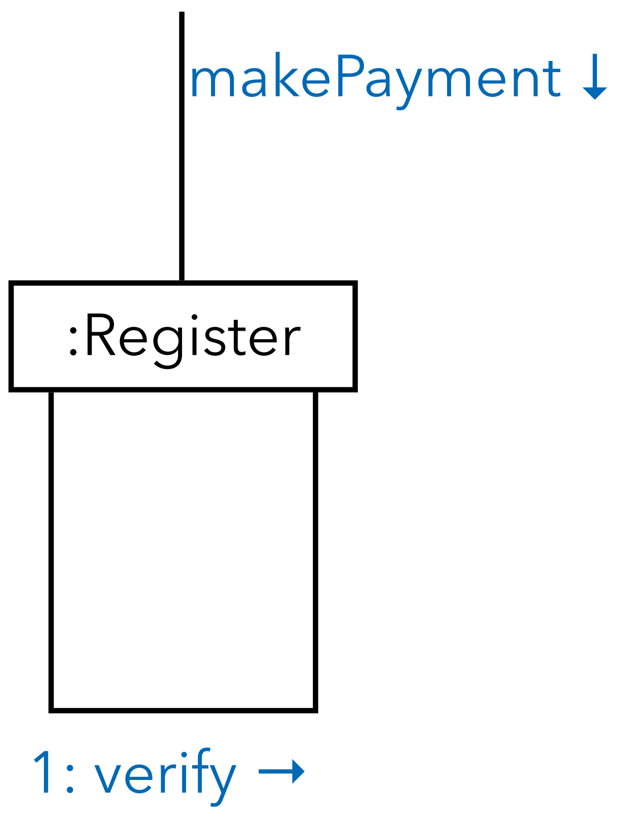
- Each **message** between objects is represented with a message expression and a small arrow indicating the direction of the message

  Sequence numbers are added to show the sequential order of messages in the current thread of control; the starting message is often not numbered.

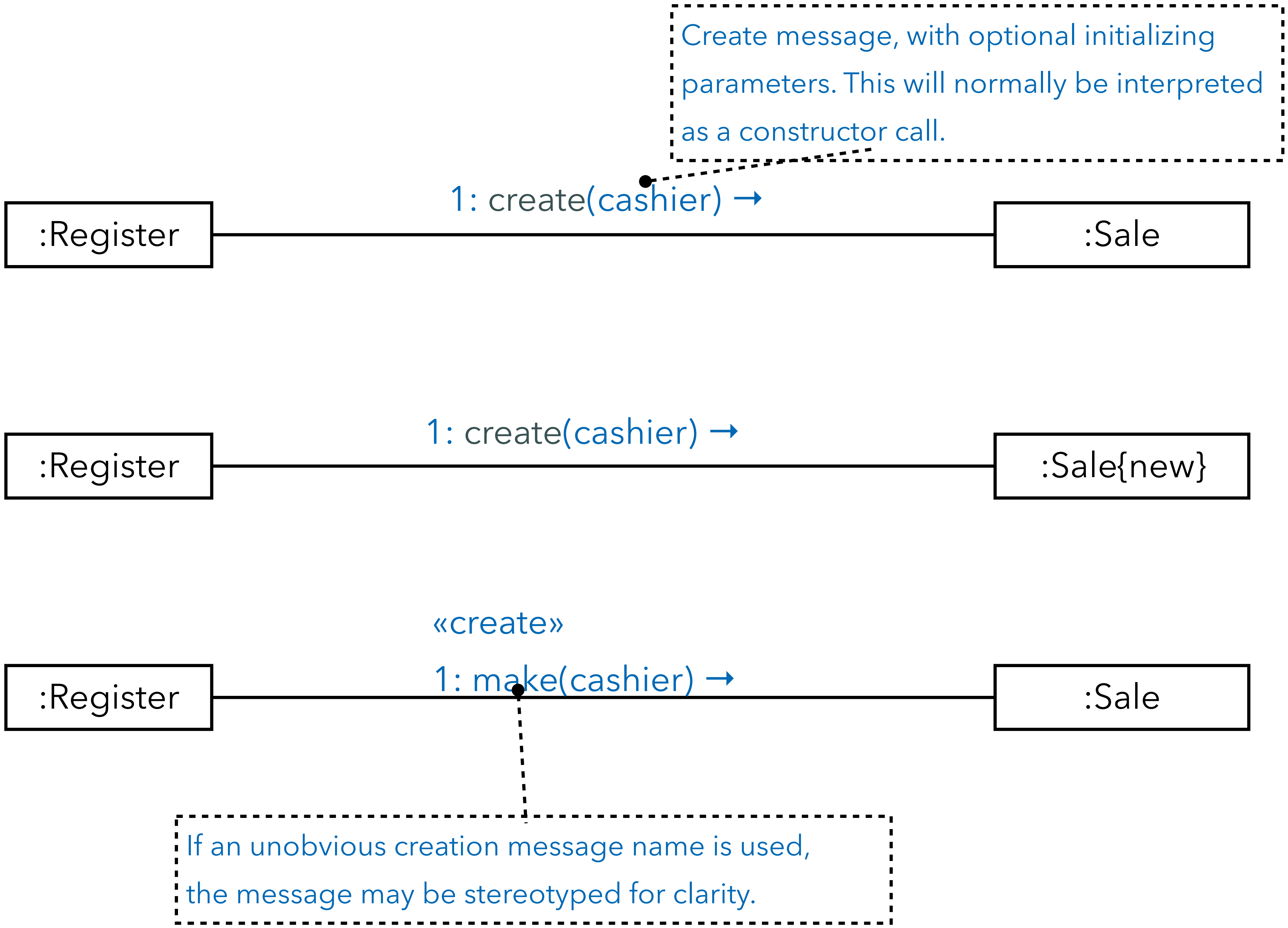


Links and Messages in Communication Diagrams

- Modeling self messages

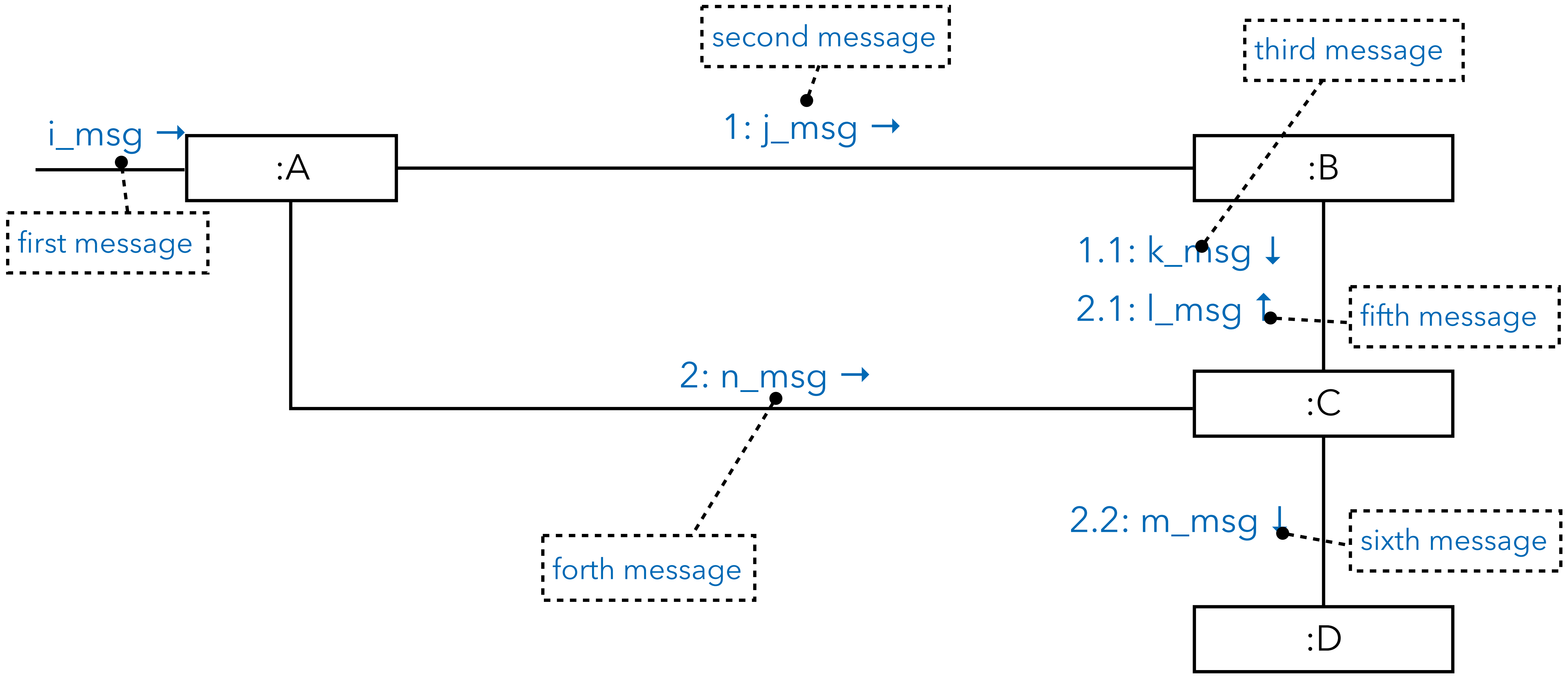


Alternative Notations for Modeling Instance Creation

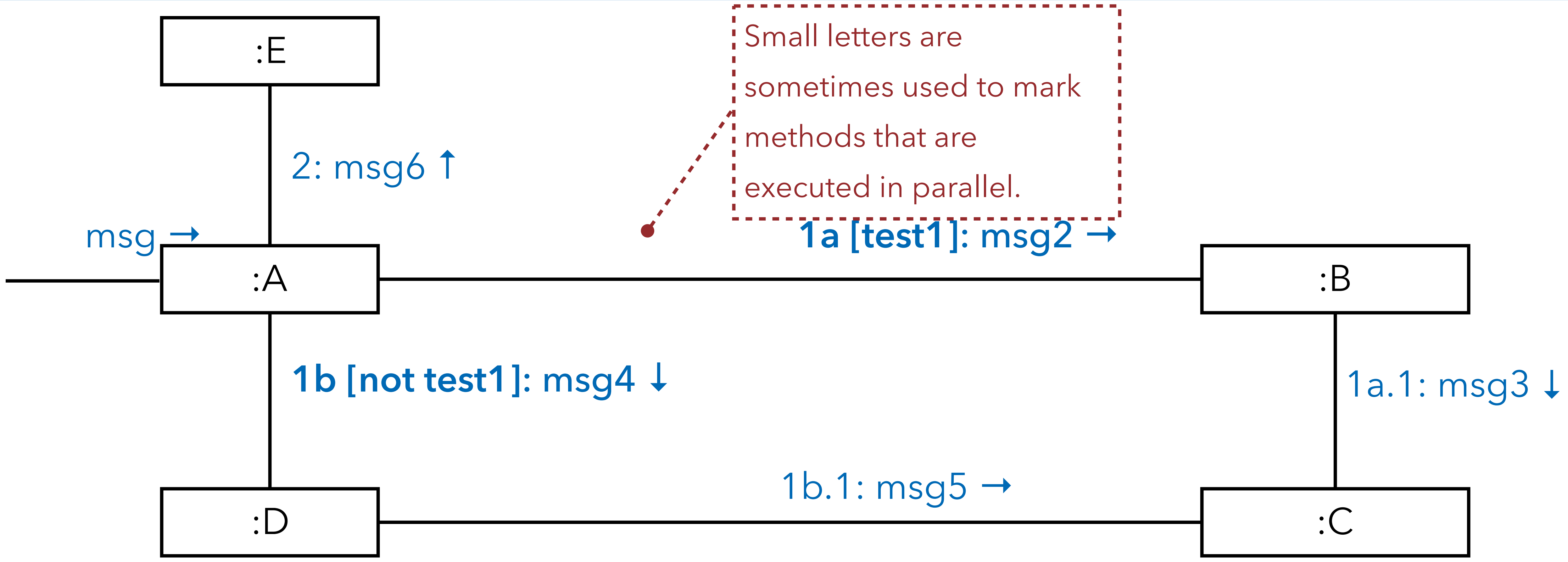

Create message, with optional initializing parameters. This will normally be interpreted as a constructor call.
1: create(cashier) →
:Register
:Sale
1: create(cashier) →
:Register
:Sale{new}
«create»
1: make(cashier) →
:Register
:Sale
If an unobvious creation message name is used, the message may be stereotyped for clarity.

Message Number Sequencing

The initial message ist not numbered to make the numbering easier to comprehend.

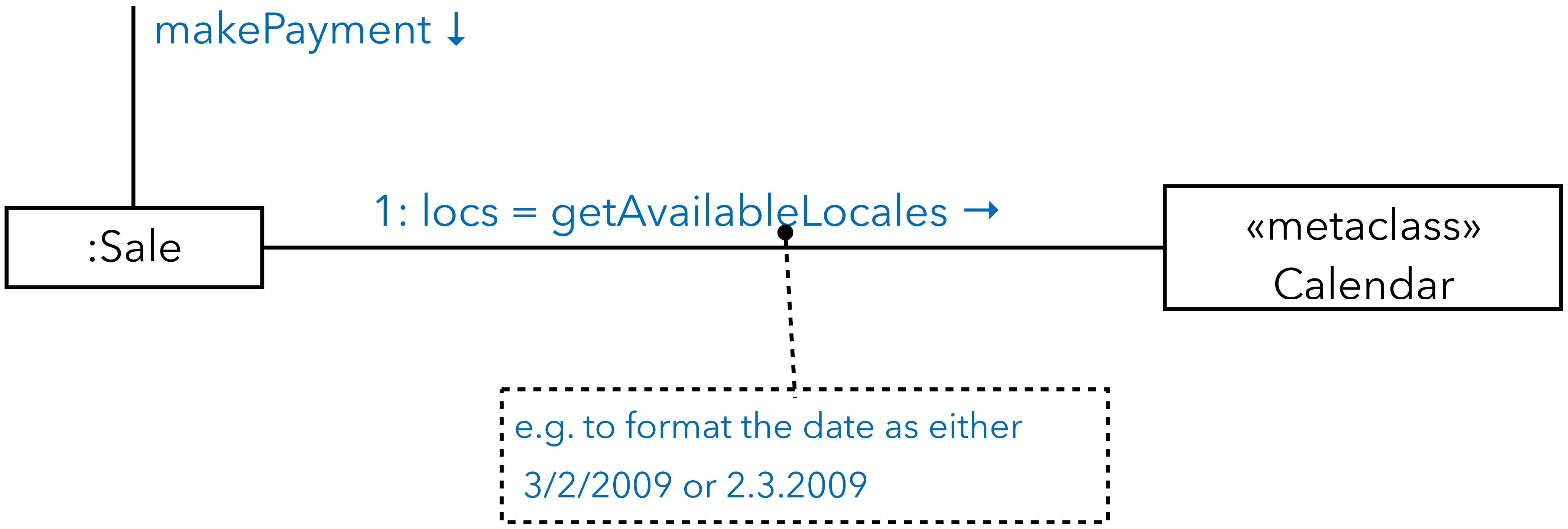


Modeling Conditional Messages



The message is only sent if the condition evaluates to true. The condition is written in square brackets. In case of modeling mutually exclusive message conditional path letters are prepended.

Messages to Class Objects


makePayment ↓
:Sale
1: locs = getAvailableLocales →
«metaclass»
Calendar
e.g. to format the date as either
3/2/2009 or 2.3.2009

UML Communication vs. UML Sequence Diagrams

TECHNISCHE UNIVERSITÄT DARMSTADT

Strengths and Weaknesses Interaction Diagrams

Type	Strengths	Weaknesses
Sequence Diagram	✓ clearly shows sequence or time ordering of messages ✓ large set of detailed notation options	– forced to extend to the right when adding new objects; consumes horizontal space
Communication Diagram	✓ space economical - flexibility to add new objects in two dimensions	– more difficult to see sequence of messages – fewer notational options

Strengths and Weaknesses Interaction Diagrams

Type	Strengths	Weaknesses
Sequence Diagram	✓ clearly shows sequence or time ordering of messages	– forced to extend to the right when adding new objects; consumes
Co Diagram	flexibility to add new objects in two dimensions	sequence of messages – fewer notational options

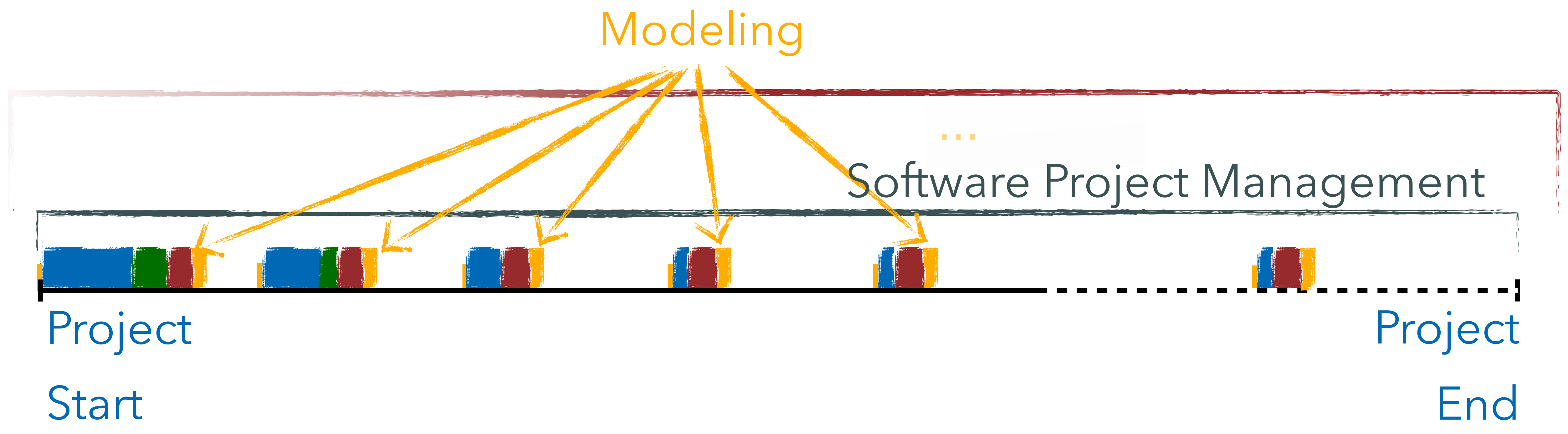
UML tools often emphasize sequence diagrams, because of their greater notational power.

Summary

The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

- Modeling the dynamic behavior is often more rewarding than modeling the static structure w.r.t. understanding a domain
- Modeling the dynamic behavior is often particularly useful if the control-flow is more involved; but only draw the part that is relevant to understand the problem at hand
- The UML is often used informally - this is OK if everyone interprets the diagrams in the same way

- The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.



🟦 Requirements Management
🟩 Domain Modeling
🟨 Modeling
🟥 Testing