Föreläsning 5
Sequence Diagrams & Design Patterns

Objektorienterad programutveckling, trådar och datakommunikation (DA343A)

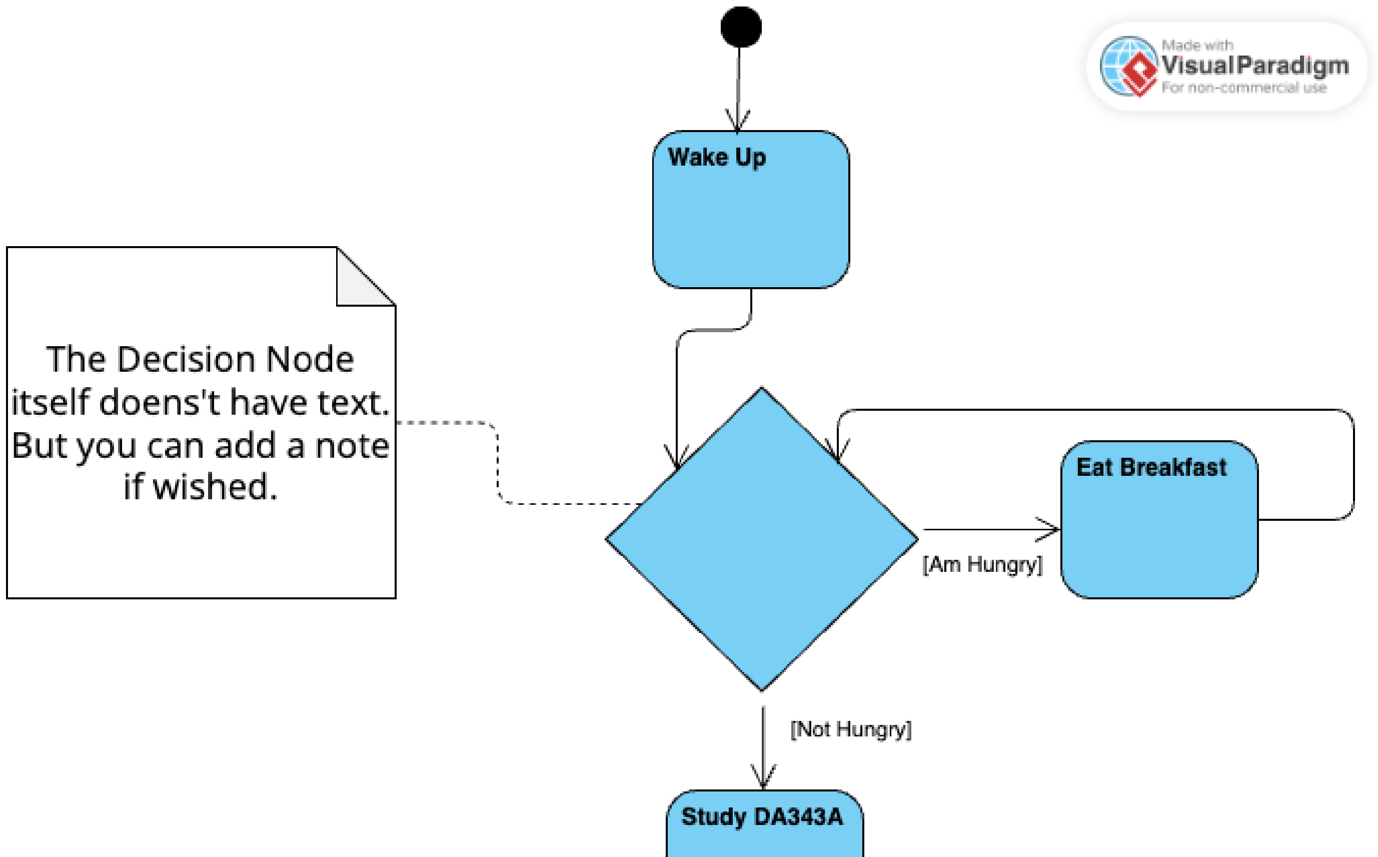
Ben Blamey

Based on Slides by Johan Holmberg.

MALMÖ UNIVERSITY

Note: Activity Diagrams

- Clarification: Decision/Merge Nodes don’t contain text.


Made with VisualParadigm For non-commercial use
Wake Up
The Decision Node itself doens't have text. But you can add a note if wished.
Eat Breakfast
[Am Hungry]
[Not Hungry]
Study DA343A

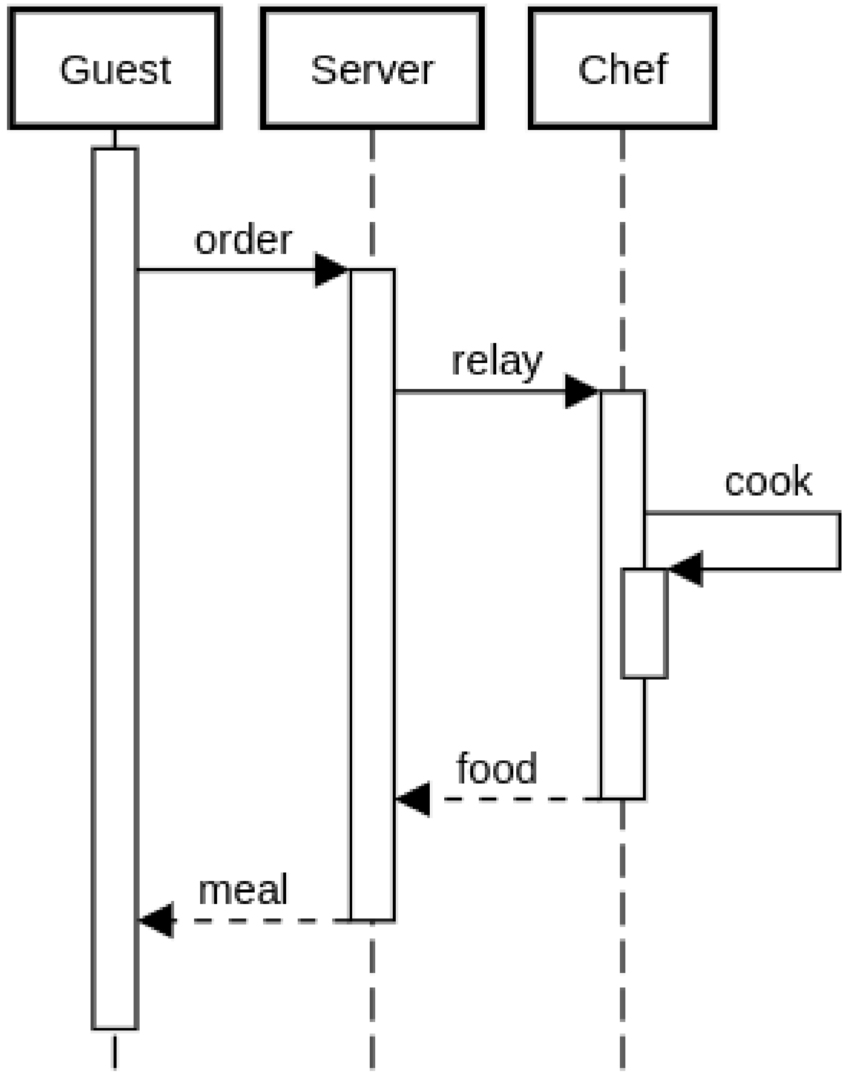
Sequence Diagram (Sekvens Diagram)

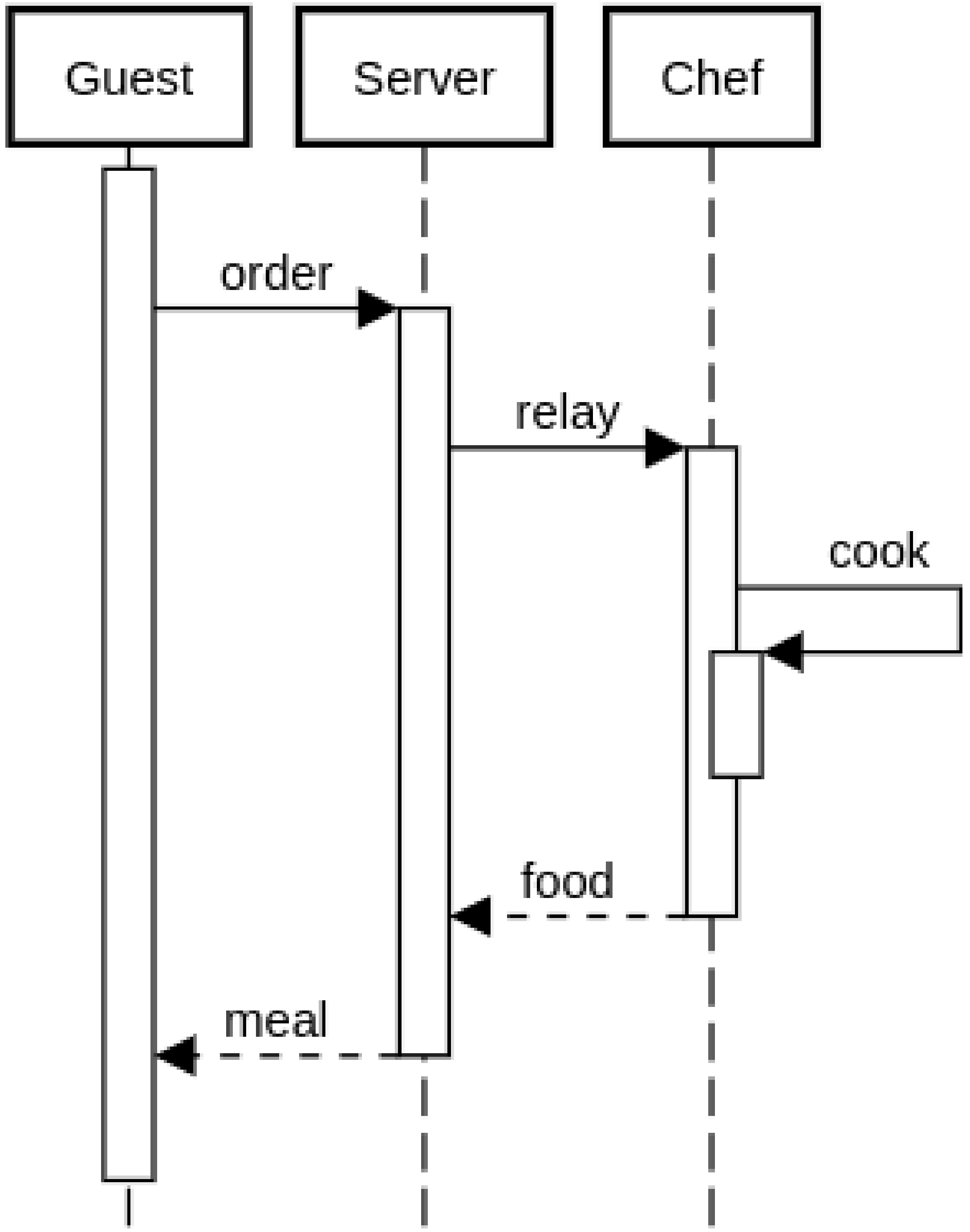
Sequence Diagram Example: Order Food in a restaurant

Guest

Server

Chef





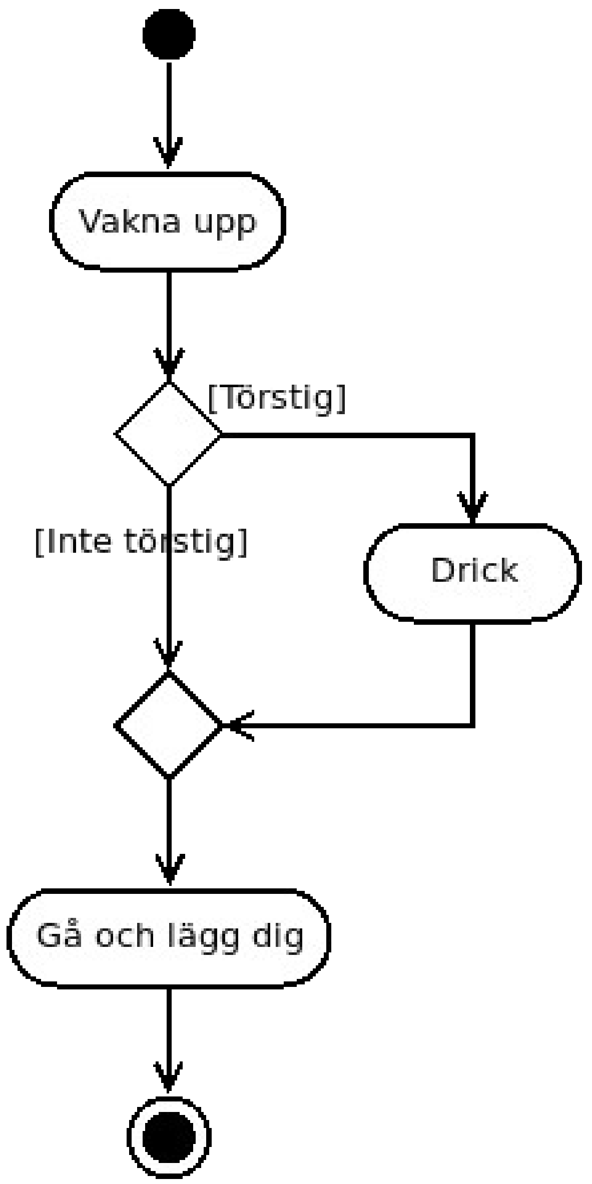
Sequence Diagram (Sekvensdiagram)

- Shows interaction between objects in a system.
- Shows which **objects** are included in the interaction, which **messages** are exchanged, and in what **order**.
- Message ordering **starts at the top.**
- Shows when the object is **active** (vertical box).

MALMÖ UNIVERSITY

Activity Diagram

From Last Lecture



- Show the flow of control in a system.
- Shows **activities** (not **objects**).

MALMÖ UNIVERSITY

Activity Diagram vs Sequence Diagram

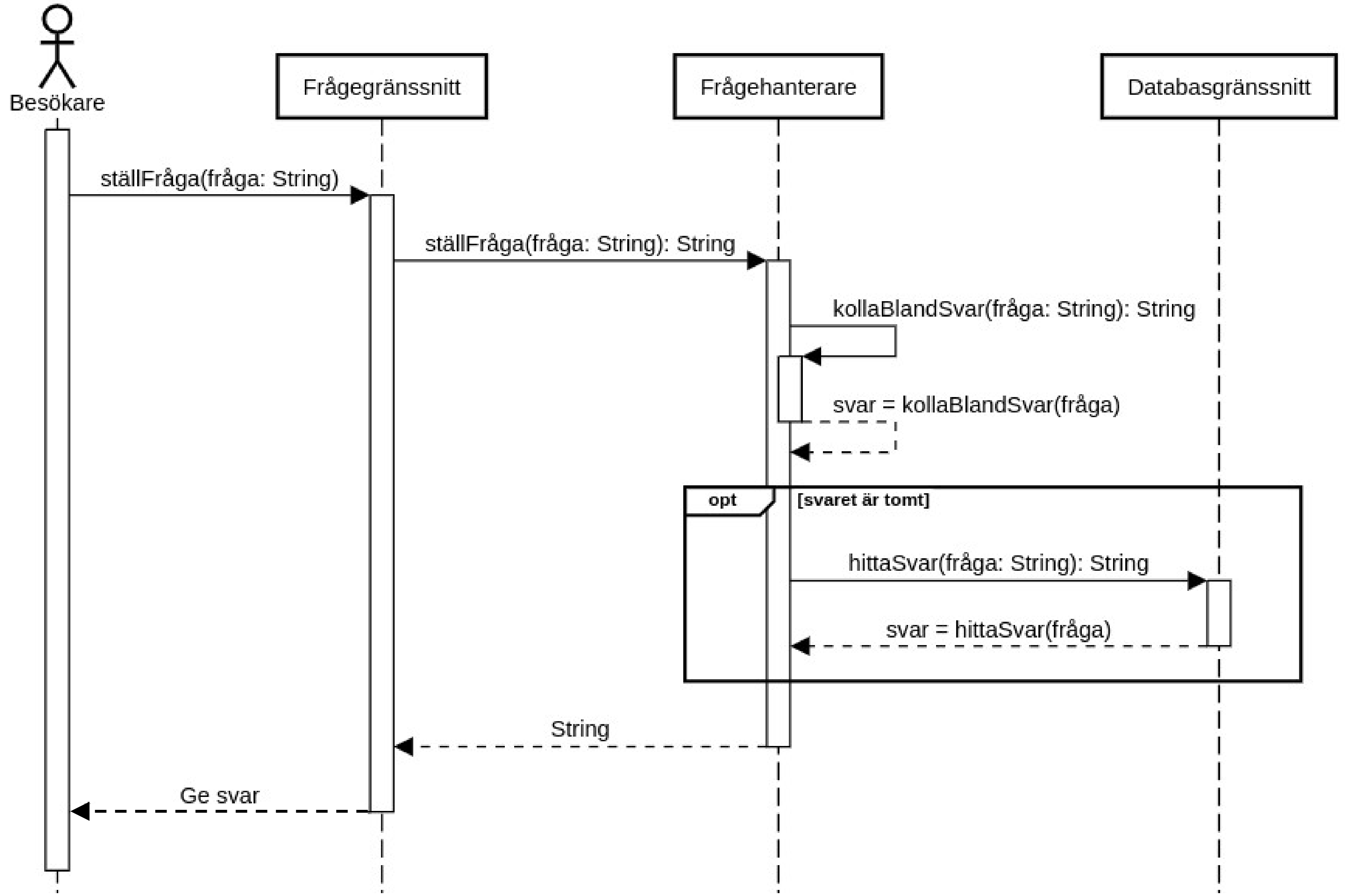
	Activity Diagram (Last Lecture)	Sequence Diagram (Today)
What does it represent?	Represents **activity flow** for a use case.	Represents communication **sequence** between **objects.**
Examples	Business Processes, Document Processes, Physical movement of real objects, etc.	Communication Protocols, Authentication Steps, detail of method calls…
Flow….	Control flow between **Activities**	Flow of **Messages**, back and forth, between objects. Ordering of messages.
Control Structures	Control structures (loops, decision nodes, concurrency)	Possible, but less common.
Diagram Layout	Choose own layout.	Objects along the top (horizontal axis), messages horizontal arrows, time flows downwards (vertical axis)

MALMÖ UNIVERSITY

Example: Reception

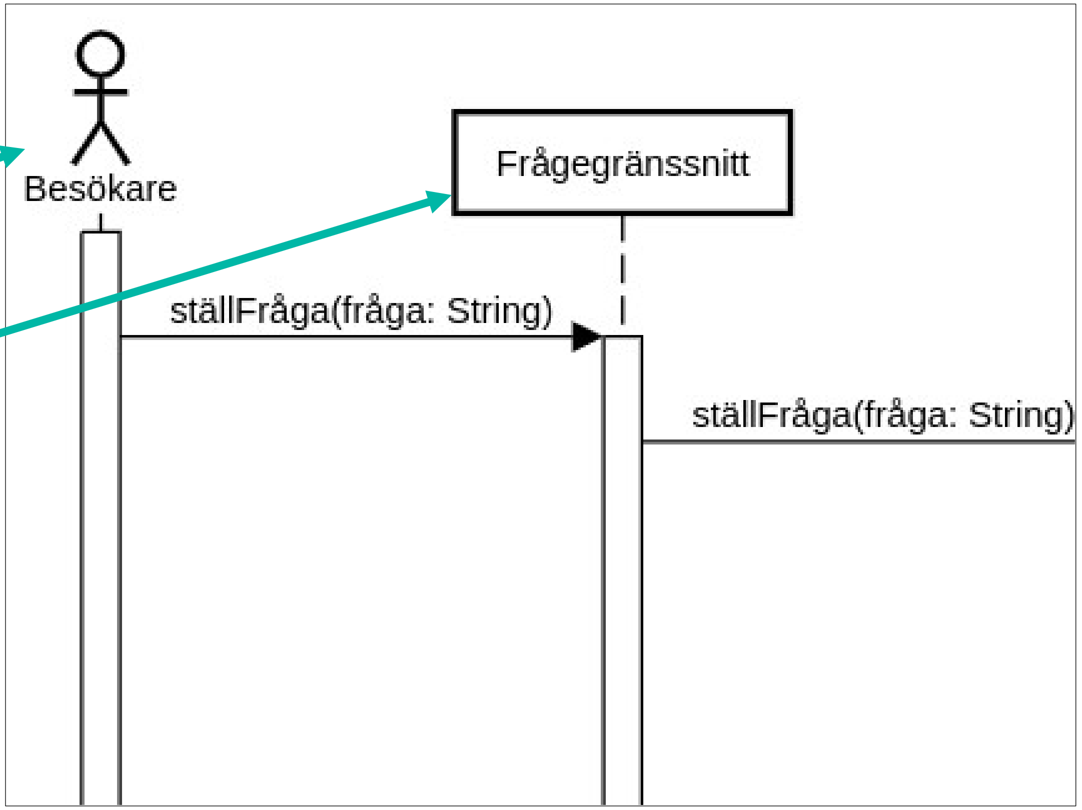
Use Case: Ask a question in the reception

Ställ en fråga (i receptionen)


Besökare
Frågegränssnitt
Frågehanterare
Databasgränssnitt
ställFråga(fråga: String)
ställFråga(fråga: String): String
kollaBlandSvar(fråga: String): String
svar = kollaBlandSvar(fråga)
opt
[svaret är tomt]
hittaSvar(fråga: String): String
svar = hittaSvar(fråga)
String
Ge svar

Sequence Diagram Components

- The figure at the top left is an actor (e.g. a user)
- The boxes represent the objects that are part of the system.

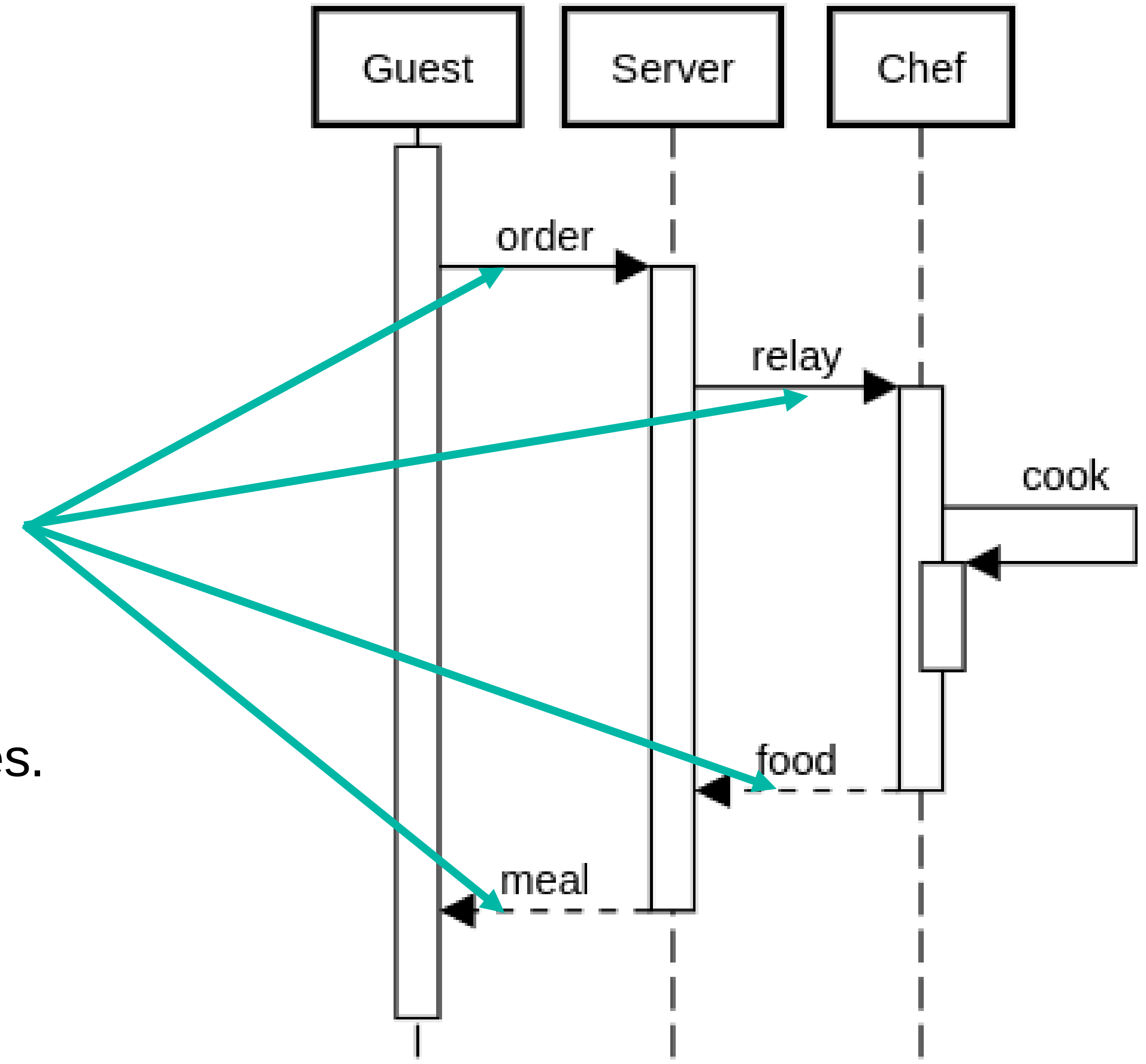


Sequence Diagram Components

The **solid arrows** represent method calls (or messages being sent).

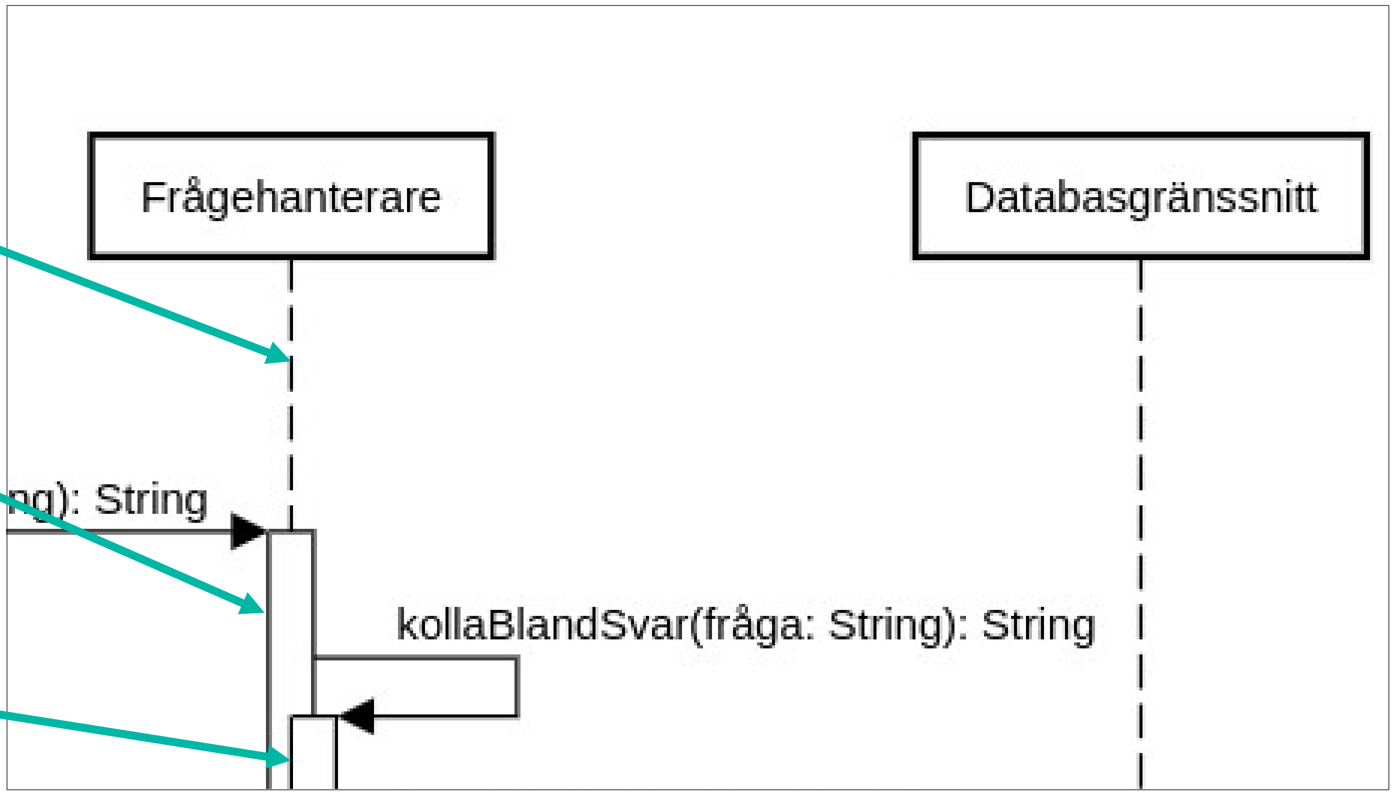
The **dashed arrows** represent method return values, (or responses).

We can add a label to explain the values.



Sequence Diagram Components

- The vertical line from the box (**lifeline**), shows the lifetime of the object.
- When an object is active, an **activity box** appears across the line.
- When an object sends a message to itself, it appears as an additional overlapping rectangle.



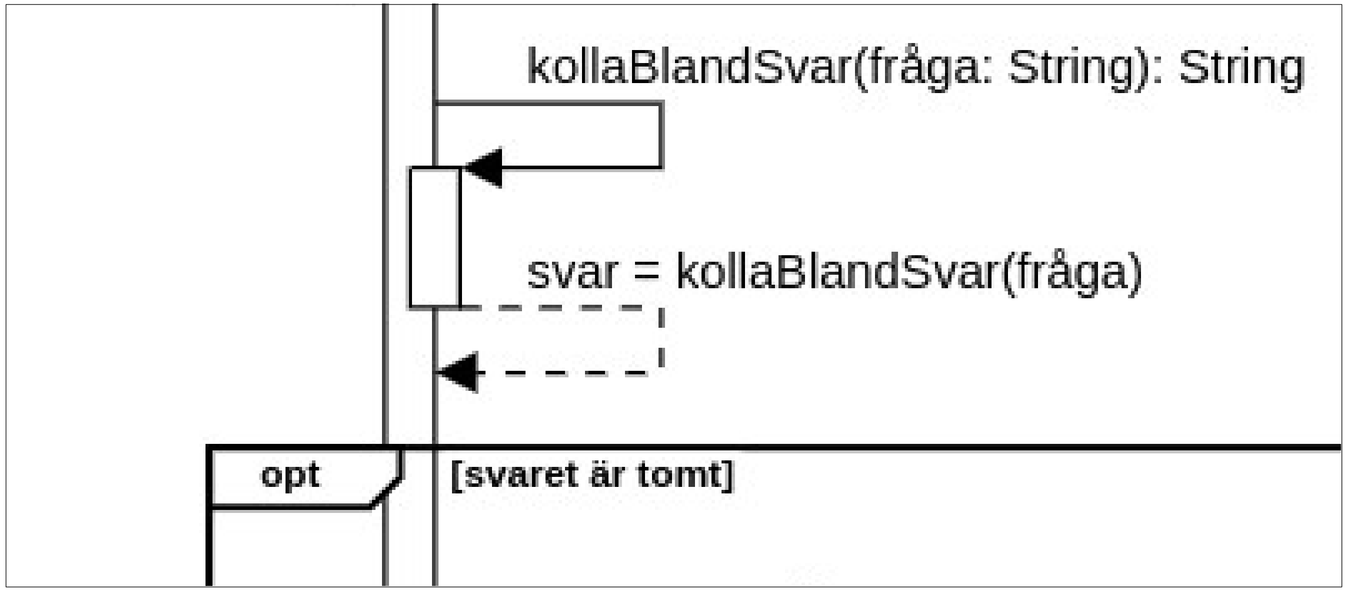
Sequence Diagram Components

Suppose we have a private method kollaBlandSvar(fråga) which checks if the front desk already knows the answer to a question.

We show

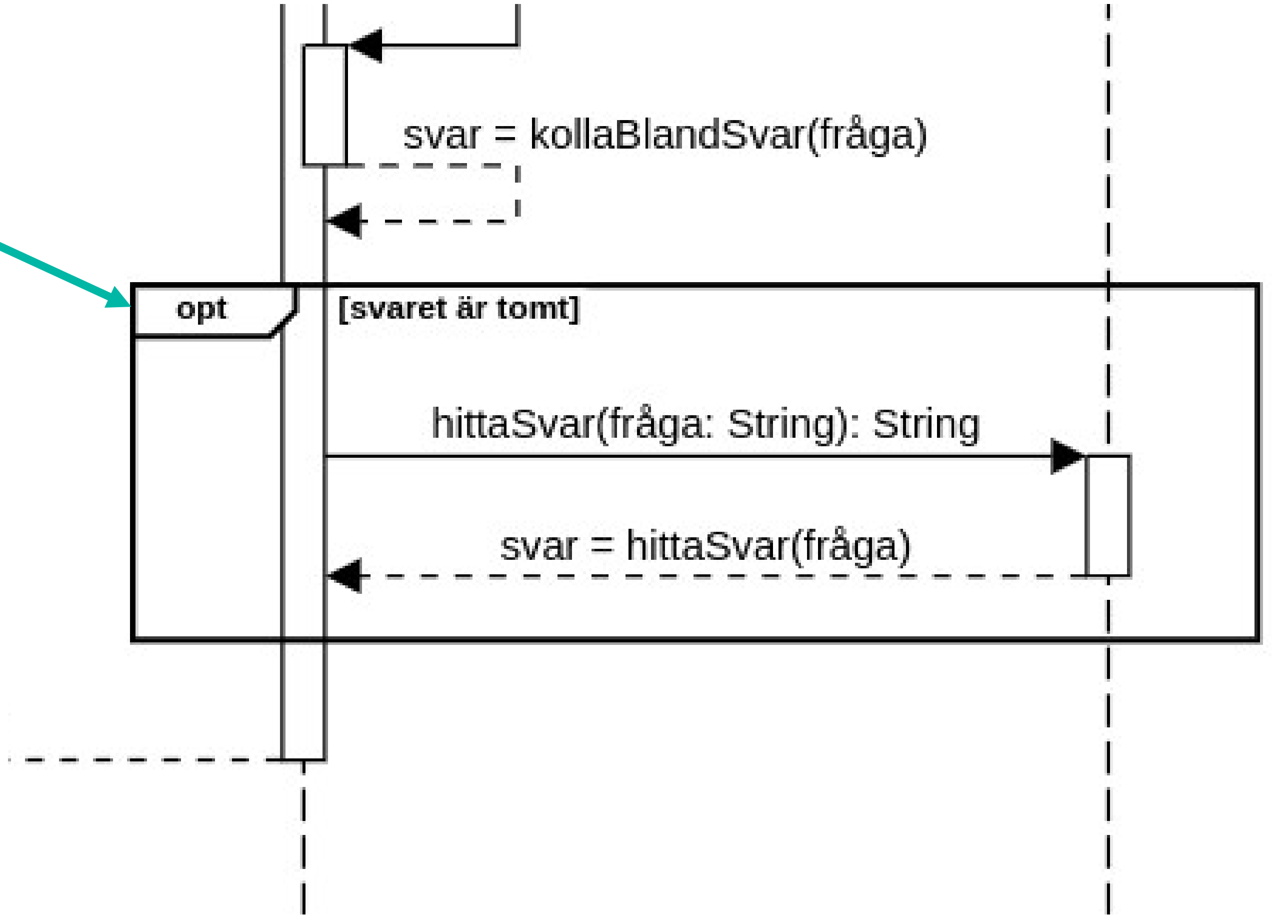
svar = kollaBlandSvar(question)

on the return (dashed) arrow.


kollaBlandSvar(fråga: String): String
svar = kollaBlandSvar(fråga)
opt
[svaret är tomt]

Sequence Diagram Components

- A **Sequence (or Interaction) Fragment** is a special region in a sequence diagram.
- It can refer to other sequence diagrams or introduce control structures.
- The **operator** in the top left indicates the type of fragment.
- All affected objects are covered by the fragment.
- Fragments contain other fragments (nesting).


svar = kollaBlandSvar(fråga)
opt
[svaret är tomt]
hittaSvar(fråga: String): String
svar = hittaSvar(fråga)

MALMÖ UNIVERSITY

Operator	Fragment Description
alt	**Alternative** multiple fragments: only the one whose condition is true will execute.
opt	**Optional**: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	**Parallel**: each fragment is run in parallel.
loop	**Loop**: the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	**Critical region**: the fragment can have only one thread executing it at once.
neg	**Negative**: the fragment shows an invalid interaction.
ref	**Reference**: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	**Sequence diagram**: used to surround an entire sequence diagram.

$\}$ We'll only use these in the exam

Source
Also see: *UML Superstructure Specification, v2.0, page 450. https://www.omg.org/spec/UML/2.0/#docs-normative-supporting*

MALMÖ UNIVERSITY

Fragment

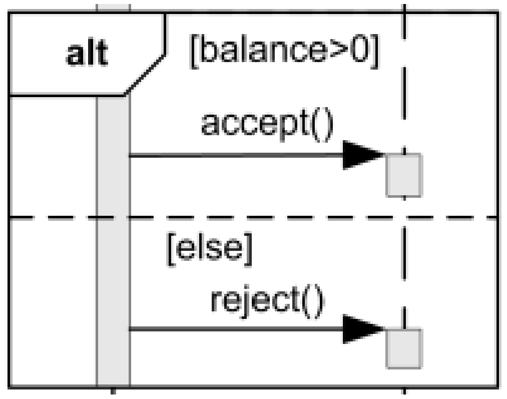
Examples:

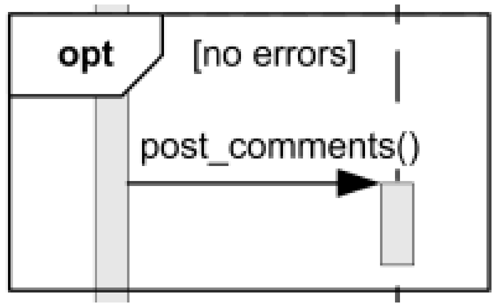
alt (alternative): an if-else statement

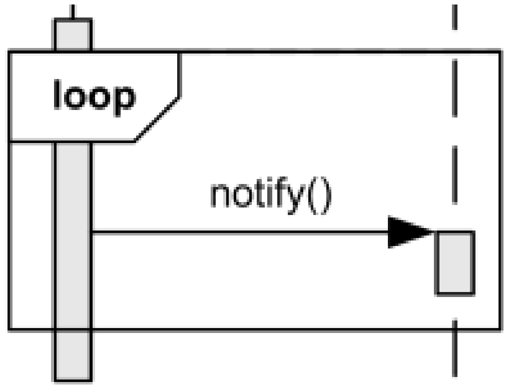
opt (optional): an IF statement

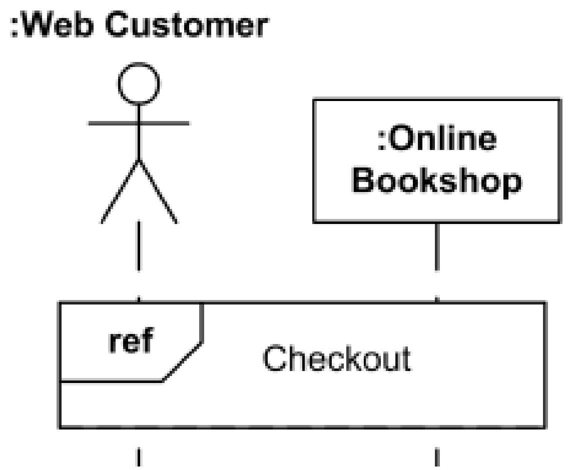
loop: A loop

ref (reference): Refer to another sequence diagram – an interaction fragment.





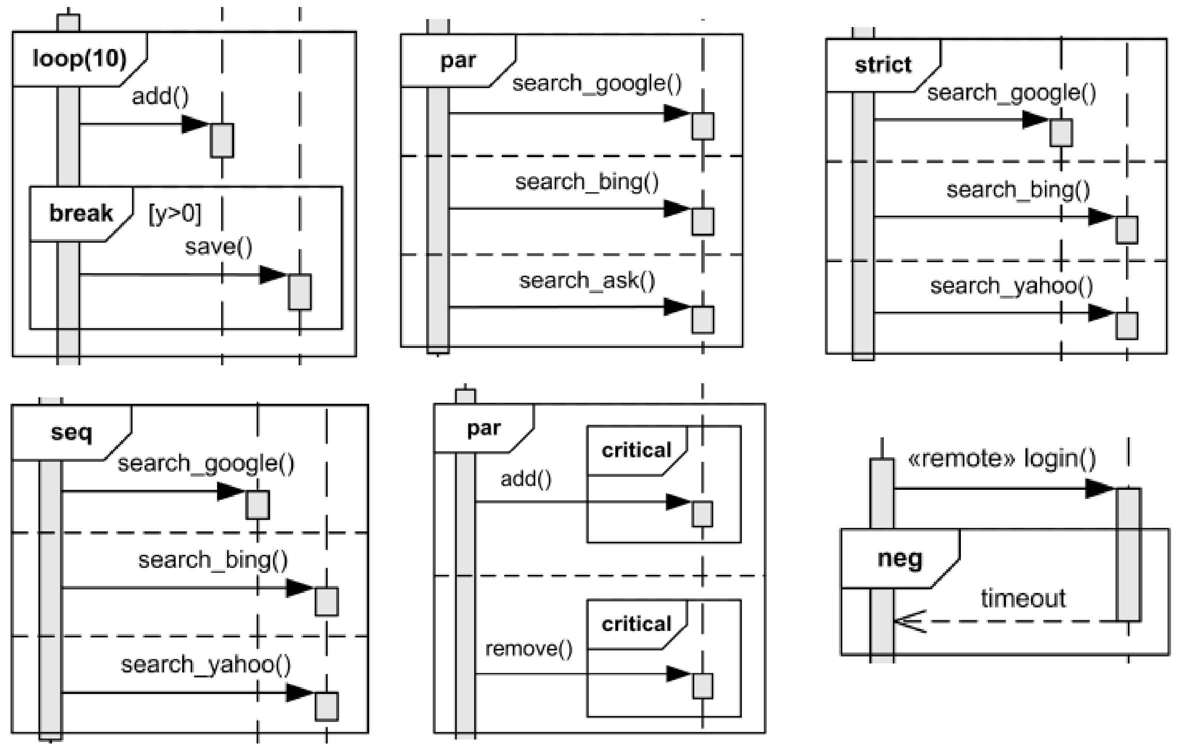




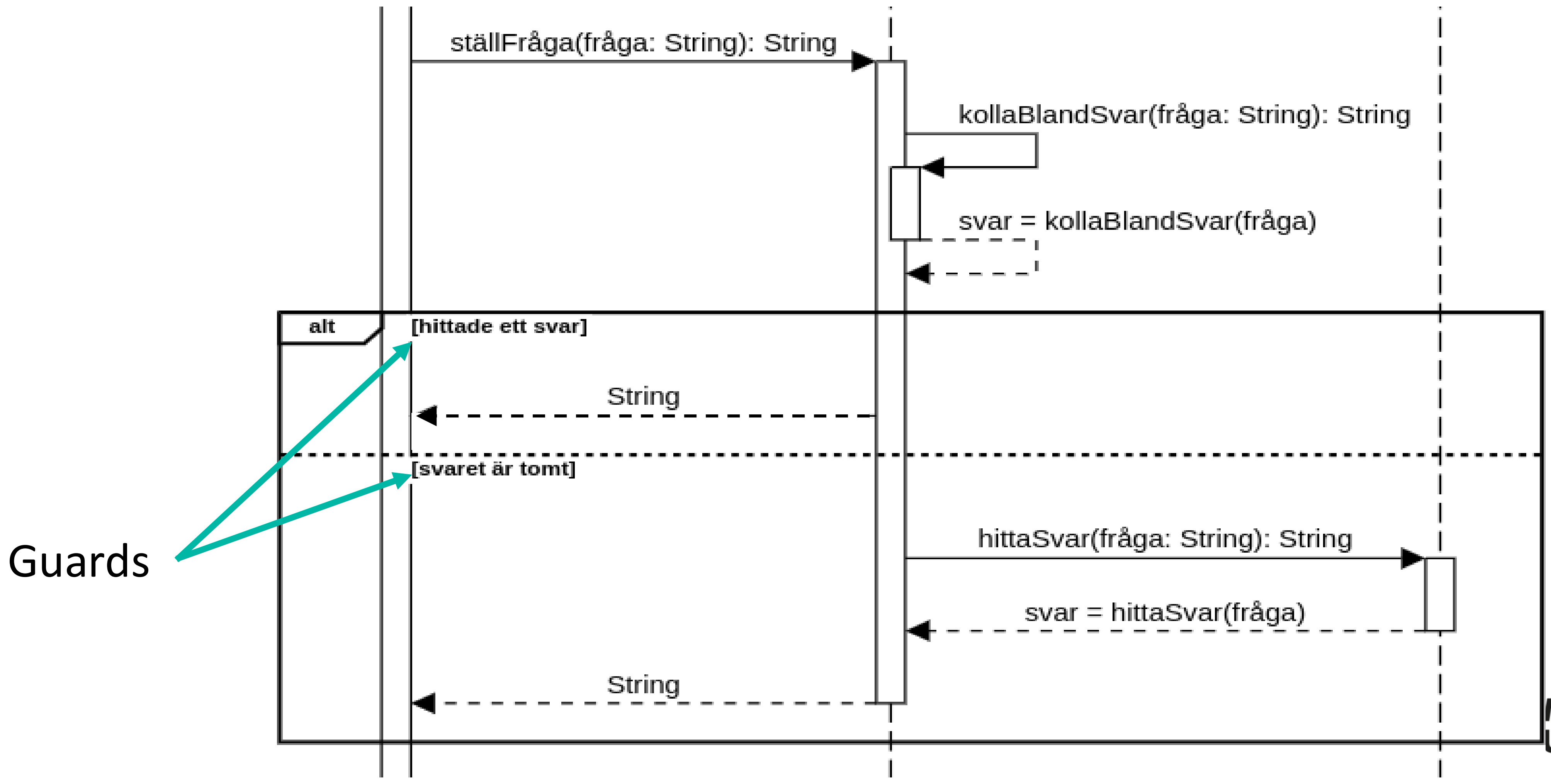
Fragment

Other fragments:

break: Break (exit) a loop.
par: Run in **parallel**.
strict: Run things in **strict** order.
seq: Run things in **sequence**.
neg: something is going wrong (negative).



Example: “alt” fragment with guards


ställFråga(fråga: String): String
kollaBlandSvar(fråga: String): String
svar = kollaBlandSvar(fråga)
alt
[hittade ett svar]
String
[svaret är tomt]
hittaSvar(fråga: String): String
svar = hittaSvar(fråga)
String
Guards

MALMÖ UNIVERSITY

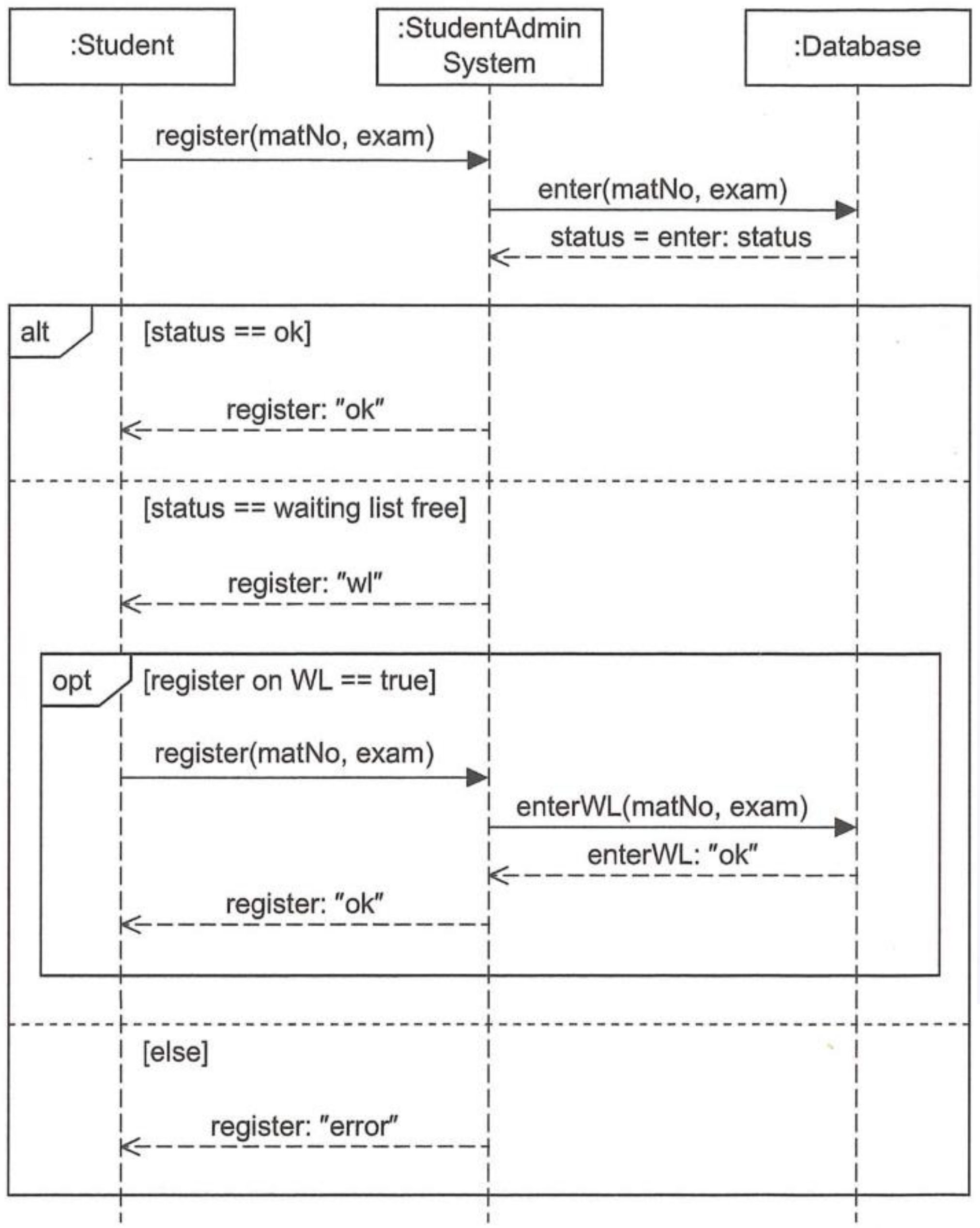


Figure 6.10
Example of an alt and an opt fragment

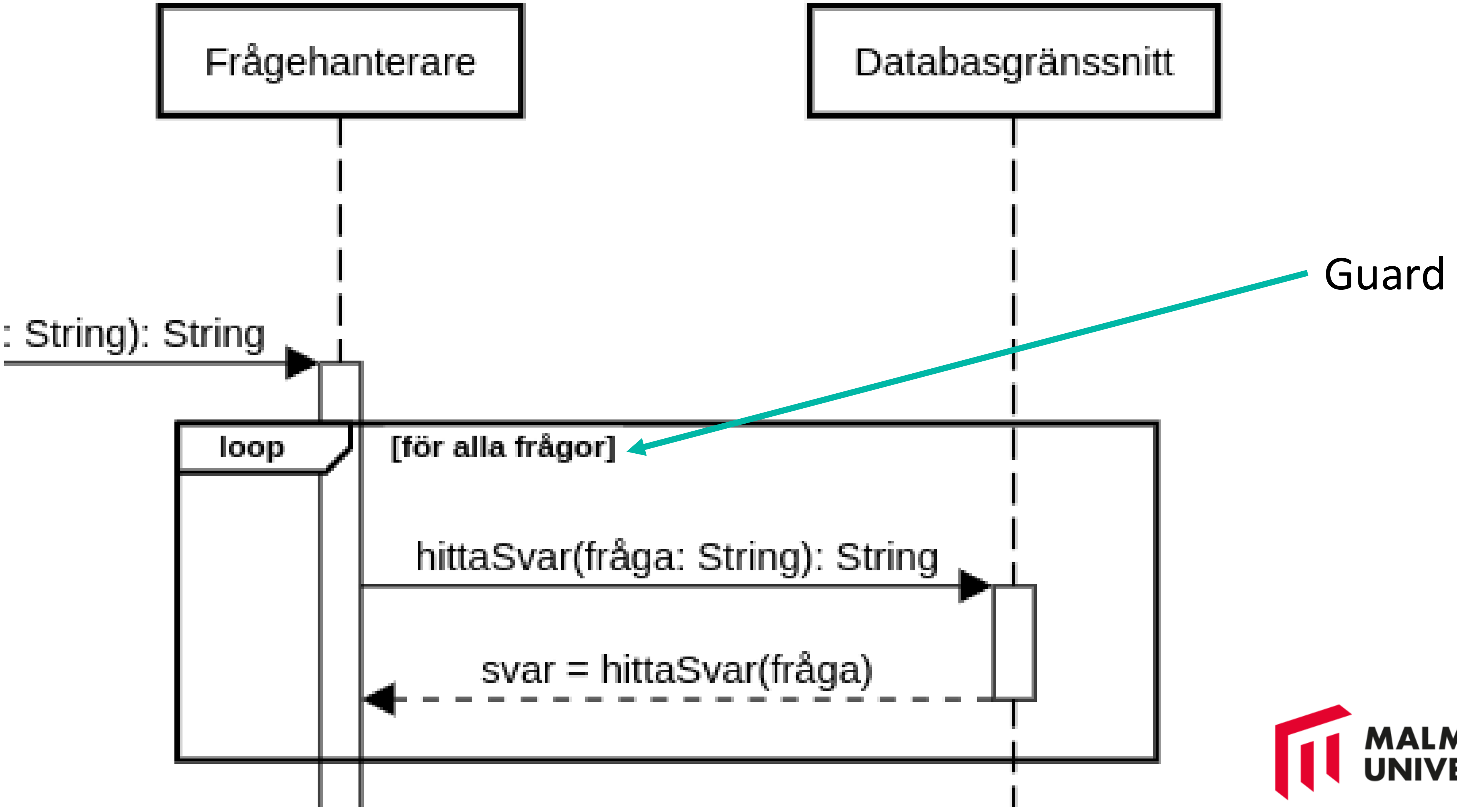
fragment. When a student wants to register for an exam, the following cases can occur: (1) There are still places available and the student can register. (2) There is a place available on the waiting list. Then the student has to decide whether to go on the waiting list. (3) If there is no place available for the exam or on the waiting list for the exam, the student receives an error message and is not registered for the course.

The opt fragment corresponds to an alt fragment with two operands, one of which is empty. The opt operator thus represents an interaction sequence whose actual execution at runtime is dependent on the guard. In a programming language, this operator would be specified as an `if` statement without an `else` branch. Figure 6.10 illustrates the use of the opt fragment. If there is a place available on the waiting list, when registering for an assignment the student can decide whether to take the

Source: UML @ Classroom, Seidl et al., page 117.
https://link.springer.com/book/10.1007/978-3-319-12742-2

UNIVERSITY

Example: “loop” fragment with guard



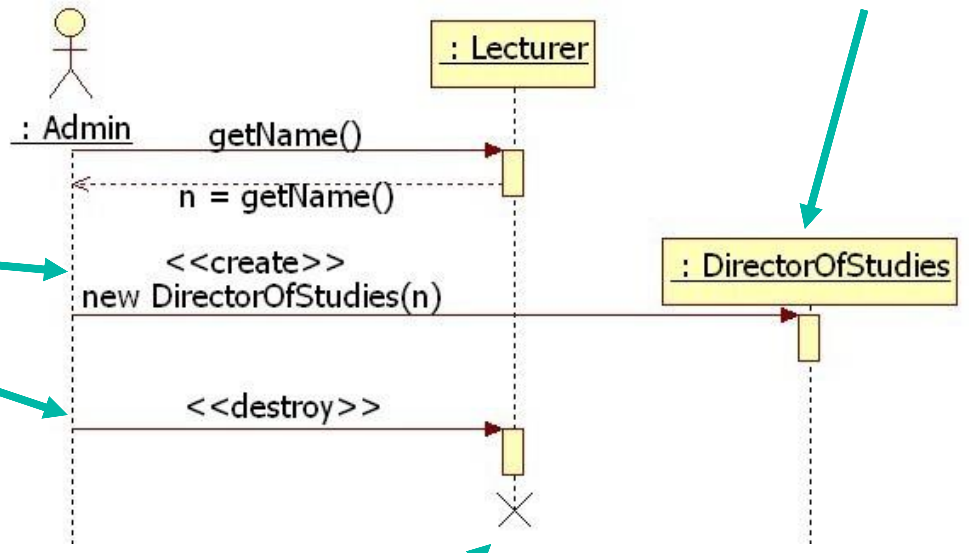
Object Creation and Destruction

We can show object creation and destruction in several ways:

(1) Stereotypes <<create>> or <<destroy>> on the arrows

(2) Showing the object lower than the others and when an object is destroyed a cross that ends the lifeline.

The object is created now



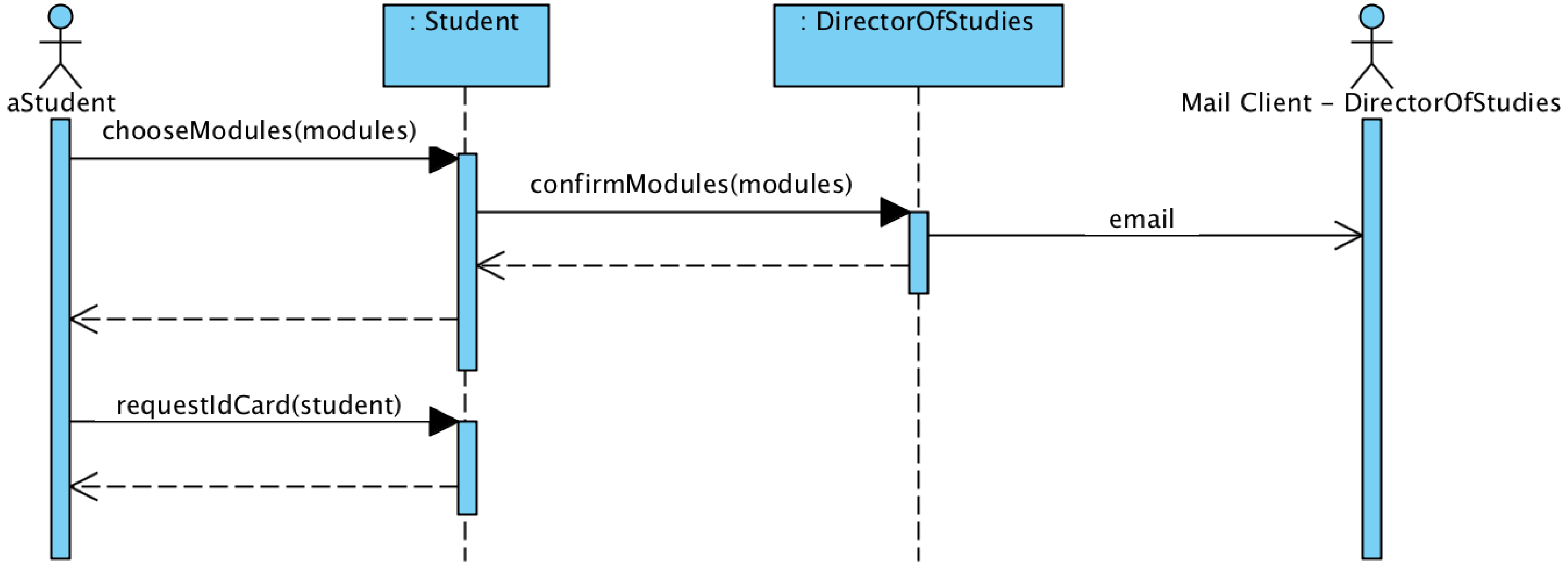
Now the object is destroyed

Asynchronous Calls

- Lines with **solid** arrow heads (——▶ or – –▶) represent **synchronous** messages/calls. Control moves to the object indicated by the arrow.
- We can show **asynchronous** messages/calls - where we don't wait for a response.
- With an **asynchronous** message, the object that sends the message keeps control, and does not wait for the object.
- An asynchronous message is displayed with an **open** arrow head (——> or – –>).
- We can use asynchronous messages when sending (or acknowledging) the message might be slow, e.g. over the Internet, or sending an email.

MALMÖ UNIVERSITY

Use case: email the director of studies



Break?

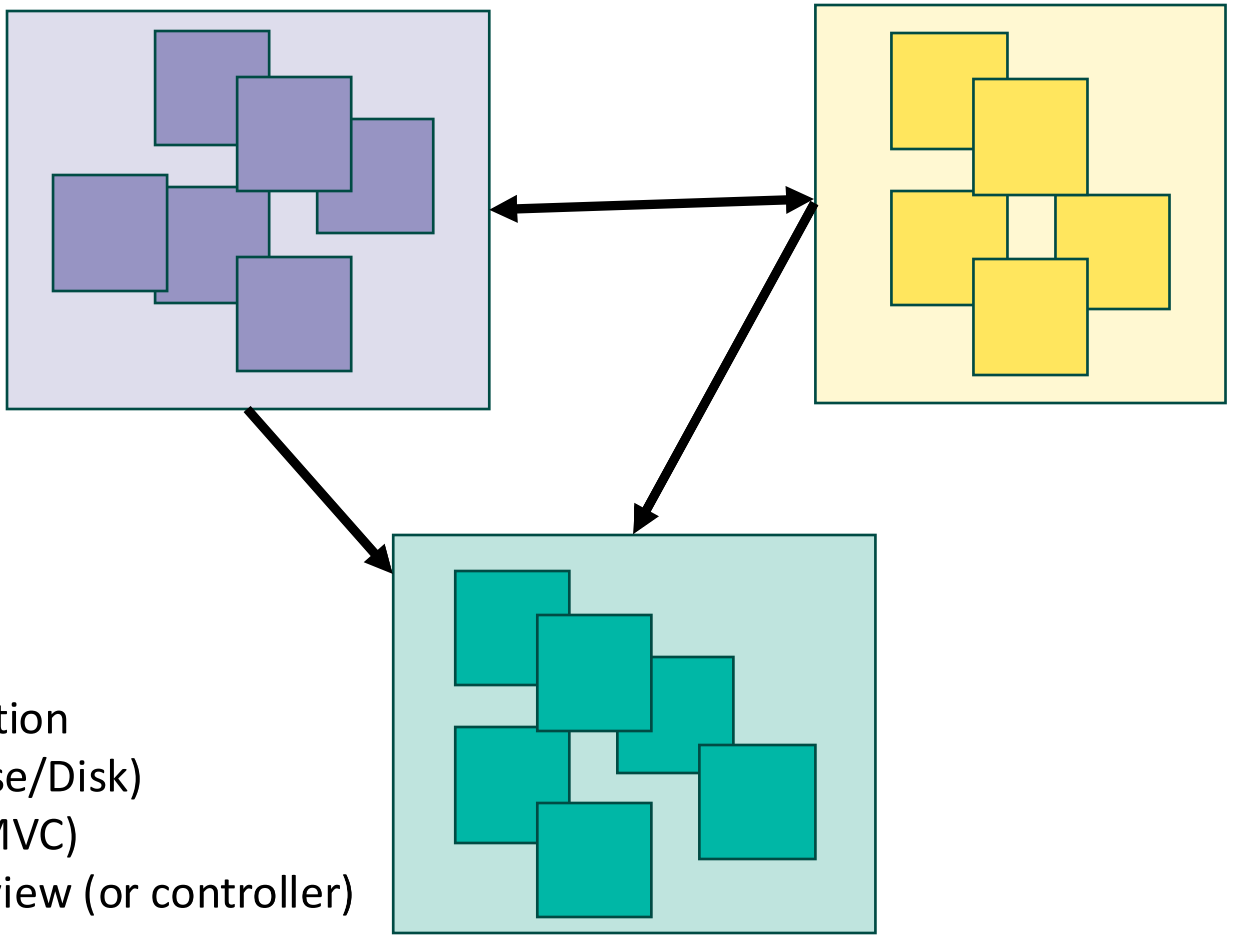
Design Patterns
(Design Mönster)

MALMÖ
UNIVERSITET

Design Patterns

- Re-usable solutions to common problems.
- Knowledge of some design patterns can help you to generate ideas.
- Design patterns serve as conventions, an can be referenced by name, to help other developers understand how the code works (and remind yourself!)
  - e.g. StringBuilder , ThingFactory, FooObserver
- The popularity of design patterns evolve over time, as practices change.
- Some design patterns are no longer considered best practice.
- Some design patterns can also be incoporated into programming languages and libraries over time.

Model View Controller (Archcitectural Pattern)



View Classes:
- User Interface
- Present Model

Controller Classes:
- Co-ordinate Models / Views

Model Classes:
- Hold state / information
- Persistence (Database/Disk)
- Business Logic (for MVC)
- No dependency on view (or controller)

- ✓ **Find** our code
- ✓ **Re-use** code (especially the model)
- ✓ Know **where** to add/extend classes
- ✓ Clear **roles** for classes (and packages). *"Separation of Concerns"*

MALMÖ UNIVERSITY

Singleton Pattern

Namn: Singleton

Problem: Ensure that only one instance of a class is created in a system.

Context: When a certain type of object should only be instantiated once in the system, but used by different parts of the system.

Implementation: achieved with a private constructor, and a static variable to give access to the single instance of the class.

No longer recommended as best practice: can creates difficulties with testing, multi-threading, extensibility. If really, really, needed, can be hidden inside a factory class.

Do not use in exams / hand-in assignments for this course!

Creational Pattern: Builder

Name: Builder

Problem: Simplifying the creation of complex objects

Context: When the creation of a certain type of object is particularly complicated and we want to separate the creation process from the actual representation of the object.

Useful in cases where the object itself is immutable (read only)

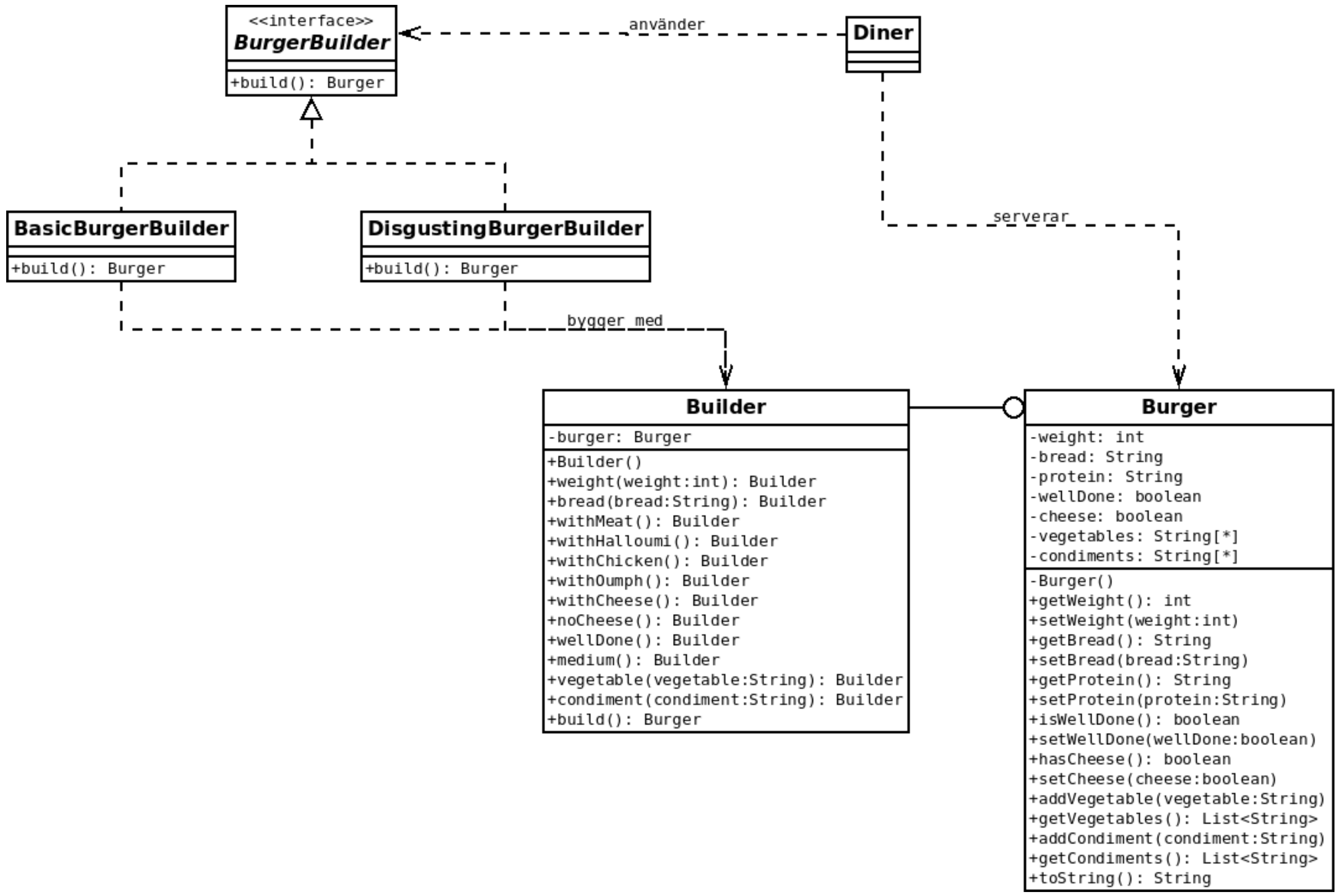
Implementation: builder object has getters/setters, and a build() method to create the final result.

Examples: StringBuilder , java.util.Locale.Builder

```
LocaleBuilder builder = new Locale.Builder()
builder.setLanguage("en")
builder.setRegion("US")
Locale locale =builder.build();
System.out.println(locale);
```

MALMÖ UNIVERSITY

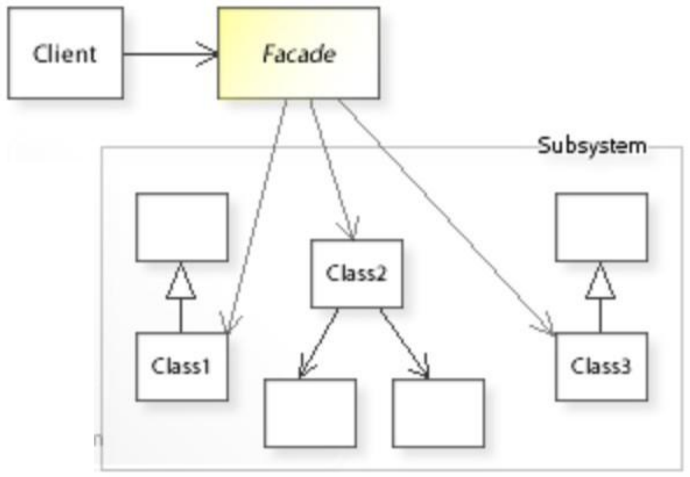
Creational Pattern: Builder

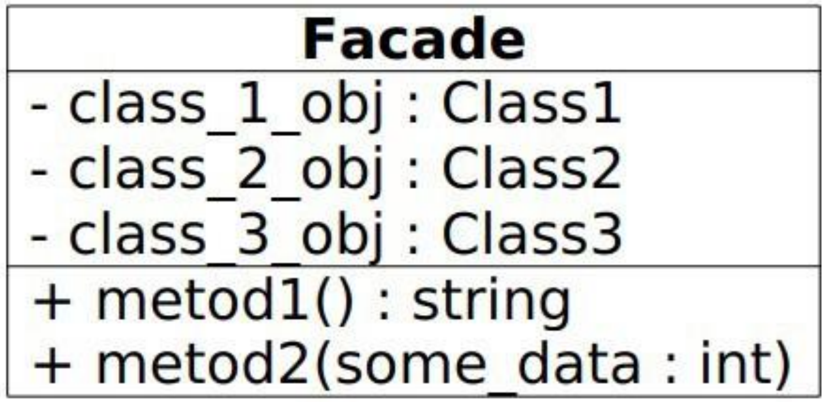


MALMÖ UNIVERSITY

Structural Pattern: Façade

- **Name**: Façade
- **Problem**: Offer an interface to a subsystem which simplifies the use of the subsystem.
- **Context**: Simplify the use of a subsystem in other systems. Hide complexity of many different classes, and present a single unified interface.
- Implementation: Instances representing the subsystem are referenced as private attributes. Public methods selectively expose the functionality.
- **Example**: (see GUI in assignment 1 JAR file).

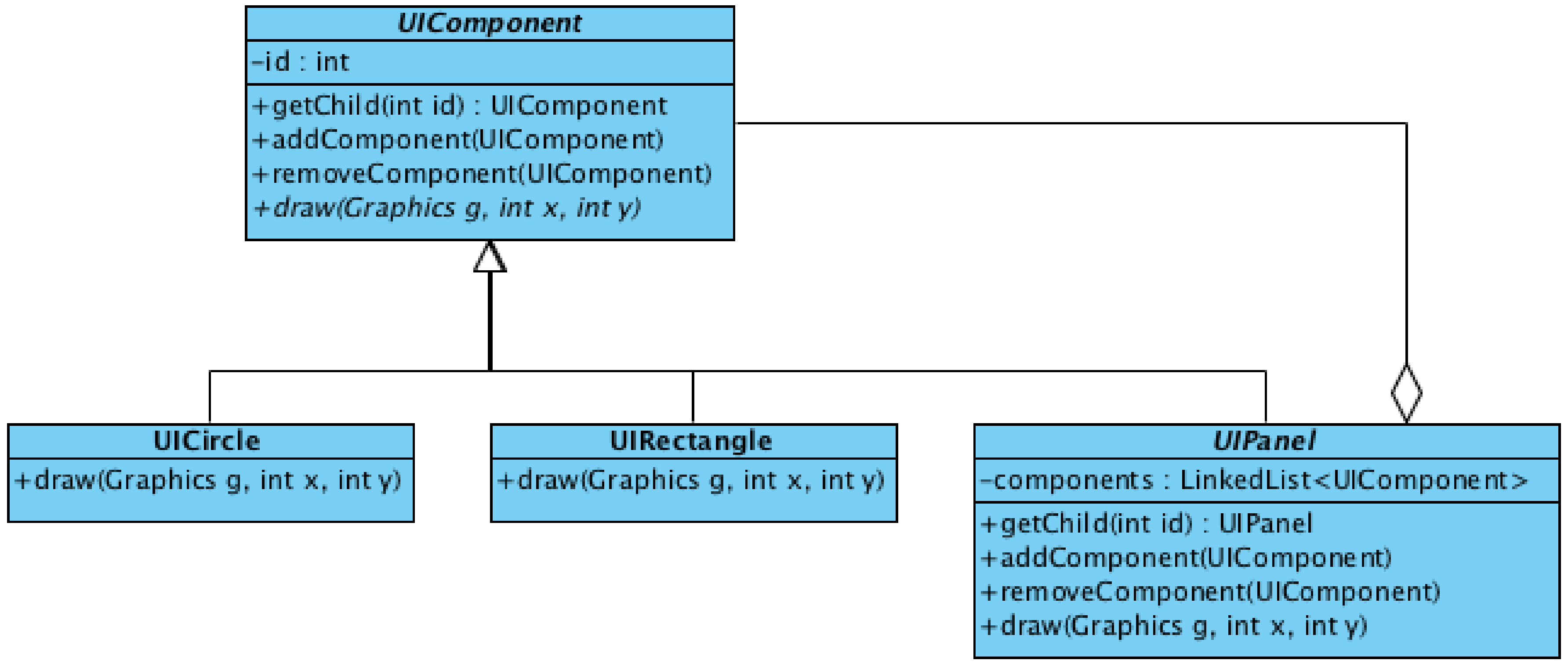




Structural Pattern: Composite

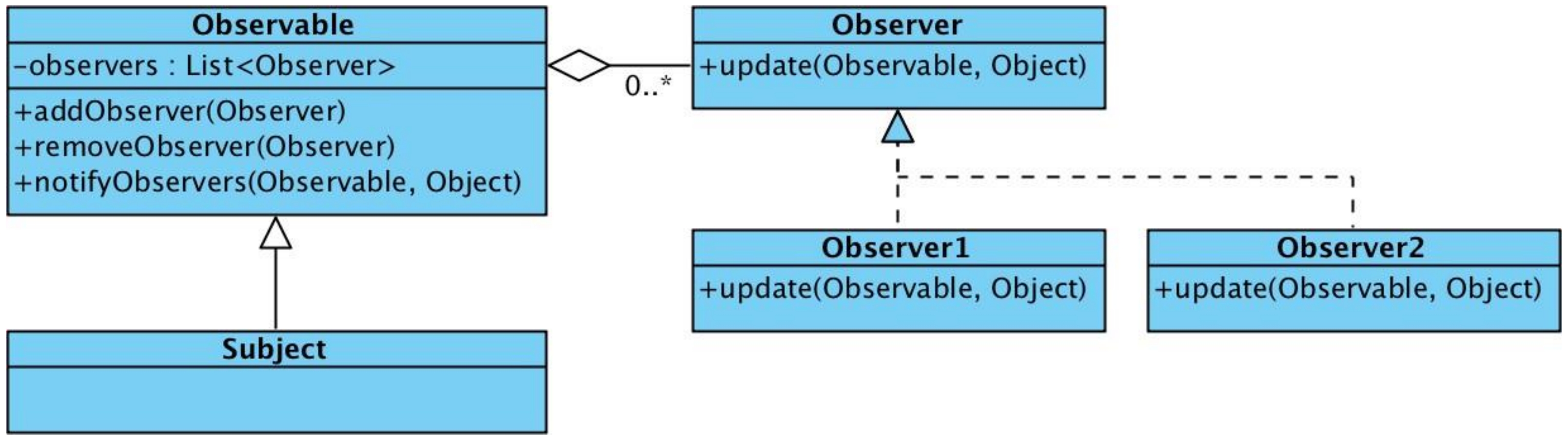
- **Name**: Composite
- **Problem**: Offering the same interface to individual objects and a collection of objects.
- **Context**: A system uses single objects and collection of objects in similar contexts.
- **Examples:** GUIs, e.g. SWING. JButton and JLabel subclass Component. Container, representing multiple components, also subclasses Component.

Structural Pattern: Composite


UIComponent
-id : int
+getChild(int id) : UIComponent
+addComponent(UIComponent)
+removeComponent(UIComponent)
+draw(Graphics g, int x, int y)
UICircle
+draw(Graphics g, int x, int y)
UIRectangle
+draw(Graphics g, int x, int y)
UIPanel
-components : LinkedList<UIComponent>
+getChild(int id) : UIPanel
+addComponent(UIComponent)
+removeComponent(UIComponent)
+draw(Graphics g, int x, int y)

Behavioural Pattern: Observer

- Name: Observer
- Problem: A change in the state of an object should be communicated to other objects.
- Context: When one or more objects should be notified when an object's state changes.
- Implementation: using callbacks (we'll cover later on)

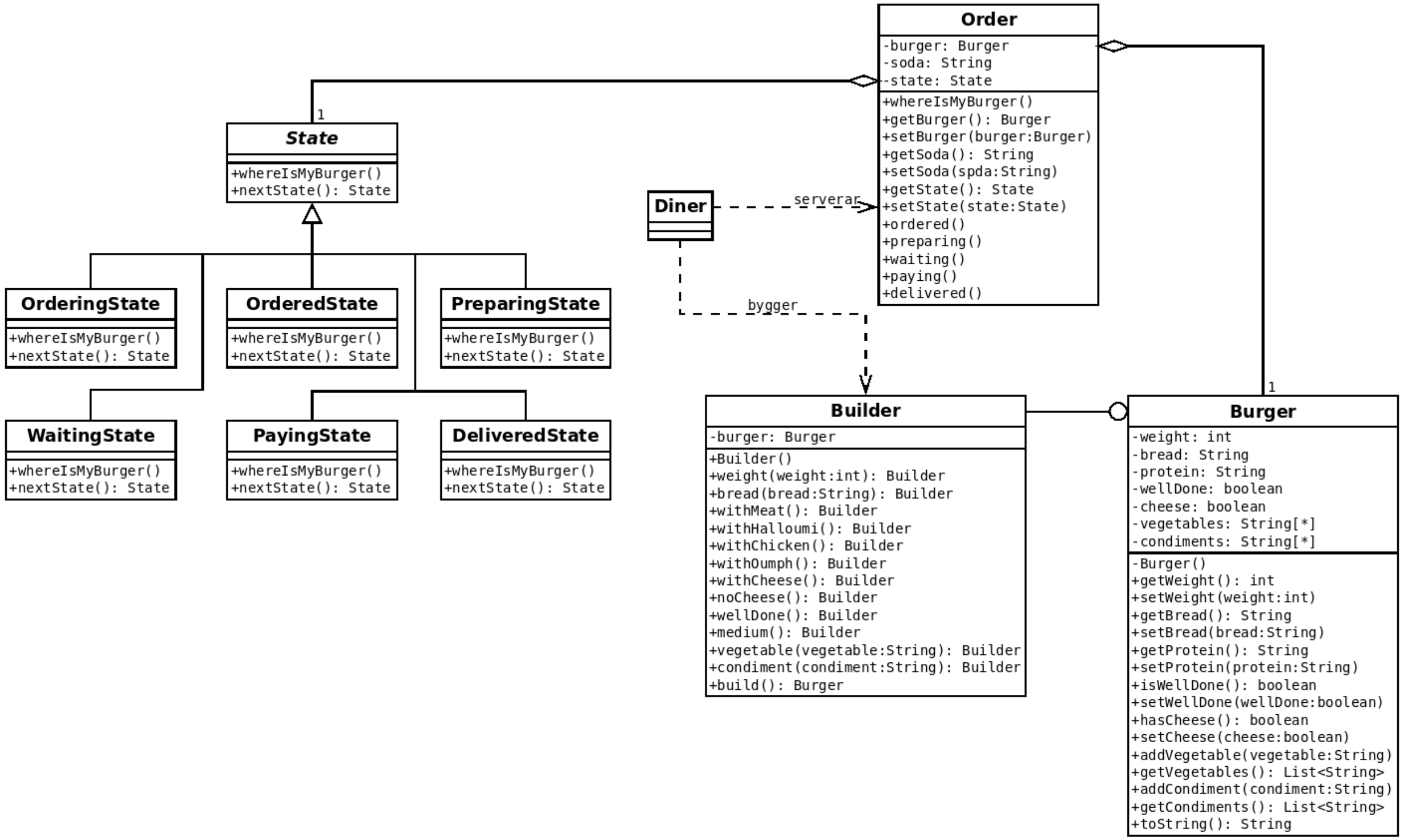

Observable
-observers : List<Observer>
+addObserver(Observer)
+removeObserver(Observer)
+notifyObservers(Observable, Object)
0..*
Observer
+update(Observable, Object)
Observer1
+update(Observable, Object)
Observer2
+update(Observable, Object)
Subject

Behavioural Pattern: State

- Name: State
- Problem: An object should have different behaviour depending on the state of the object.
- Context: Depending on the state of the object, a message to the object produces different behaviour.
- Examples: network communication (e.g. disconnected/connected).
- Implementation: hierarchy of classes representing each state (and implementation of their behaviour).
- (In simple cases, might use a private enum (or even Boolean) and switch statements.)
- Related to the concept of State Machines.

MALMÖ UNIVERSITY

Behavioural Pattern: State



Questions

MALMÖ UNIVERSITY

Reading Recommendations

Object-Oriented Systems Analysis and Design Using UML

- 9.1-3 Sekvensdiagram
- 15 Designmönster

UML Diagrams.org

- https://www.uml-diagrams.org/sequence-diagrams.html