

USG Innotiv – Professionals in ICT

The Object Oriented Concept



usg innotiv
professionals in ict

onvoorwaardelijk betrouwbaar

Objectives

- Explain how a software design may be represented as a set of interacting objects that manage their own state and operations
- Describe the activities in the object-oriented design process
- Introduce various models that can be used to describe an object-oriented design
- Show how the UML may be used to represent these models



Agenda

- Object-oriented, the hype
- Objects and classes
- Generalisation and inheritance
- Designing your project
- Additional example
- Object-oriented in SAP
- Summary



Object-oriented, the hype.

- What are object-oriented (OO) methods?
 - OO methods provide a set of techniques for analysing, decomposing, and modularising software system architectures
 - Object-oriented analysis, design and programming are related but distinct.
 - In general, OO methods are characterized by structuring the system architecture on the basis of its *objects* (and classes of objects) rather than the *actions* it performs
- What is the rationale for using OO?
 - In general, systems evolve and functionality changes, but objects and classes tend to remain stable over time
 - Use it for **large systems**
 - Use it for **systems that change often**



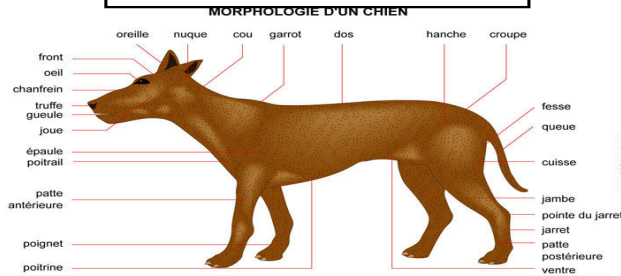
Characteristics of Object Oriented Design

- Objects are abstractions of real-world or system entities and manage themselves.
- Objects are independent and encapsulate state and representation information.



Characteristics of Object Oriented Design

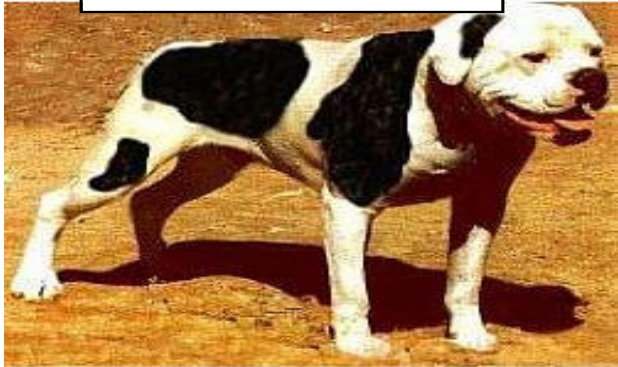
Dog Class



Dog ::Terrier



Dog ::Pitbull



Dog ::Dalmatian



Advantages of OOD

- Easier maintenance. Objects may be understood as stand-alone entities.
- Objects are potentially reusable components.
- For some systems, there may be an obvious mapping from real world entities to system objects.



Agenda

- Object-oriented, the hype
- **Objects and classes**
- Generalisation and inheritance
- Designing your project
- Additional example
- Object-oriented in SAP
- Summary



Objects and classes

- **Objects** are entities in a software system which represent instances of real-world and system entities.
- **Classes** are templates for objects. They may be used to create objects.
- Classes may **inherit** attributes and services from other classes.



Objects and classes

*An **object** is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.*

*Objects are created according to some **class** definition. An class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.*



Object identification

- Identifying **objects** (or classes) is the most difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.



Approaches to identification

- Use a grammatical approach based on a natural language description of the system (used in OOD method).
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.



Agenda

- Object-oriented, the hype
- Objects and classes
- **Generalisation and inheritance**
- Designing your project
- Additional example
- Object-oriented in SAP
- Summary

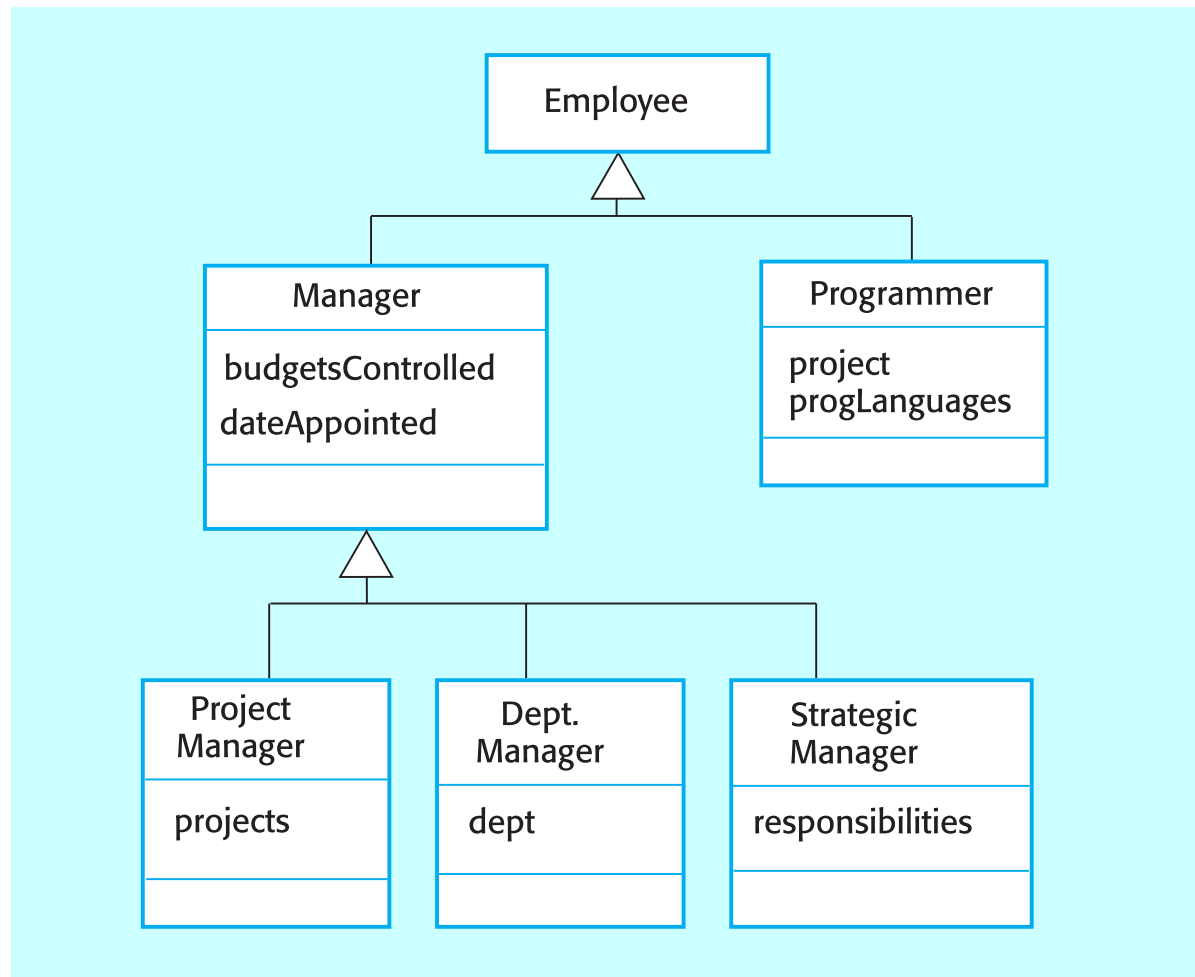


Generalisation and inheritance

- **Objects** are members of classes that define attribute types and operations.
- **Classes** may be arranged in a class hierarchy where one class (a super-class) is a generalisation of another class (sub-class).
- A **sub-class** inherits the attributes and operations from its super class and may add new methods or attributes of its own.
- Generalisation in the UML is implemented as inheritance in OO programming languages.
- Allows polymorphism



A generalisation hierarchy



Advantages of inheritance

- It is an **abstraction** mechanism which may be used to classify entities.
- It is a **reuse** mechanism at both the design and the programming level.
- The inheritance graph is a source of organisational **knowledge**.



Problems with inheritance

- Classes are **not self-contained**. They cannot be understood without reference to their super-classes.
- Designers have a tendency to reuse the inheritance graph created during analysis. It can lead to significant **inefficiency**.
- The inheritance graphs of analysis, design and implementation have different functions and should be **separately maintained**.



Agenda

- Object-oriented, the hype
- Objects and classes
- Generalisation and inheritance
- **Designing your project**
- Additional example
- Object-oriented in SAP
- Summary



Designing your project the *OO* way

- Design models
- Design process
- Association model
- Use case model
- The Class Diagram
- The Collaboration Diagram
- Sequence model



Design models

- Design models show the objects and classes and relationships between these entities.
- Static models describe the static structure of the system in terms of classes and relationships.
- Dynamic models describe the dynamic interactions between objects.



Examples of design models

- Sub-system models that show logical groupings of objects into coherent subsystems.
- Sequence models that show the sequence of object interactions.
- State machine models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalisation models, etc.



Designing your project the *OO* way

- Design models
- **Design process**
- Association model
- Use case model
- The Class Diagram
- The Collaboration Diagram
- Sequence model



An object-oriented design process

- Structured design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an essential communication mechanism.



Process stages

- Highlights key activities without being tied to any proprietary process.
 - Define the context and modes of use of the system;
 - Design the system architecture;
 - Identify the principal system objects;
 - Develop design models;
 - Specify object interfaces.

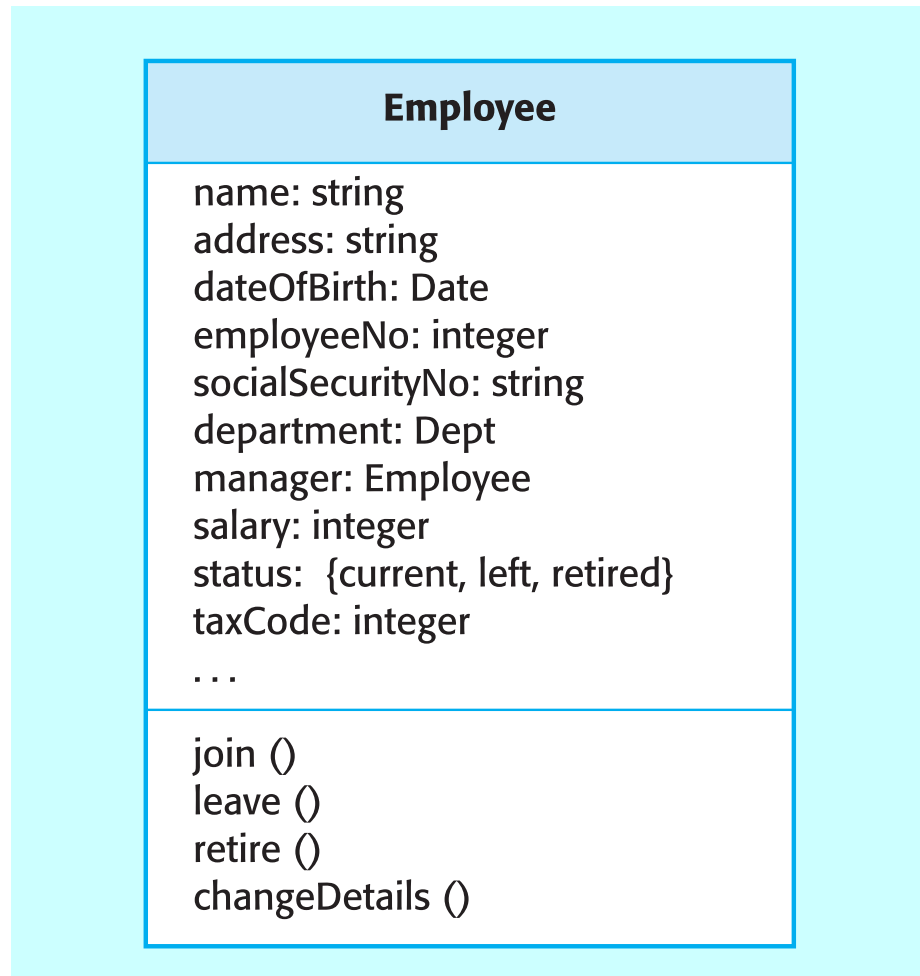


The Unified Modeling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.
- The Unified Modeling Language is an integration of these notations.
- It describes notations for a number of different models that may be produced during OO analysis and design.
- It is now a *de facto* standard for OO modelling.



Employee class (UML)



UML associations

- Objects and classes participate in relationships with other objects and classes.
- In the UML, a generalised relationship is indicated by an association.
- Associations may be annotated with information that describes the association.
- Associations are general but may indicate that an attribute of an object is an associated object or that a method relies on an associated object.

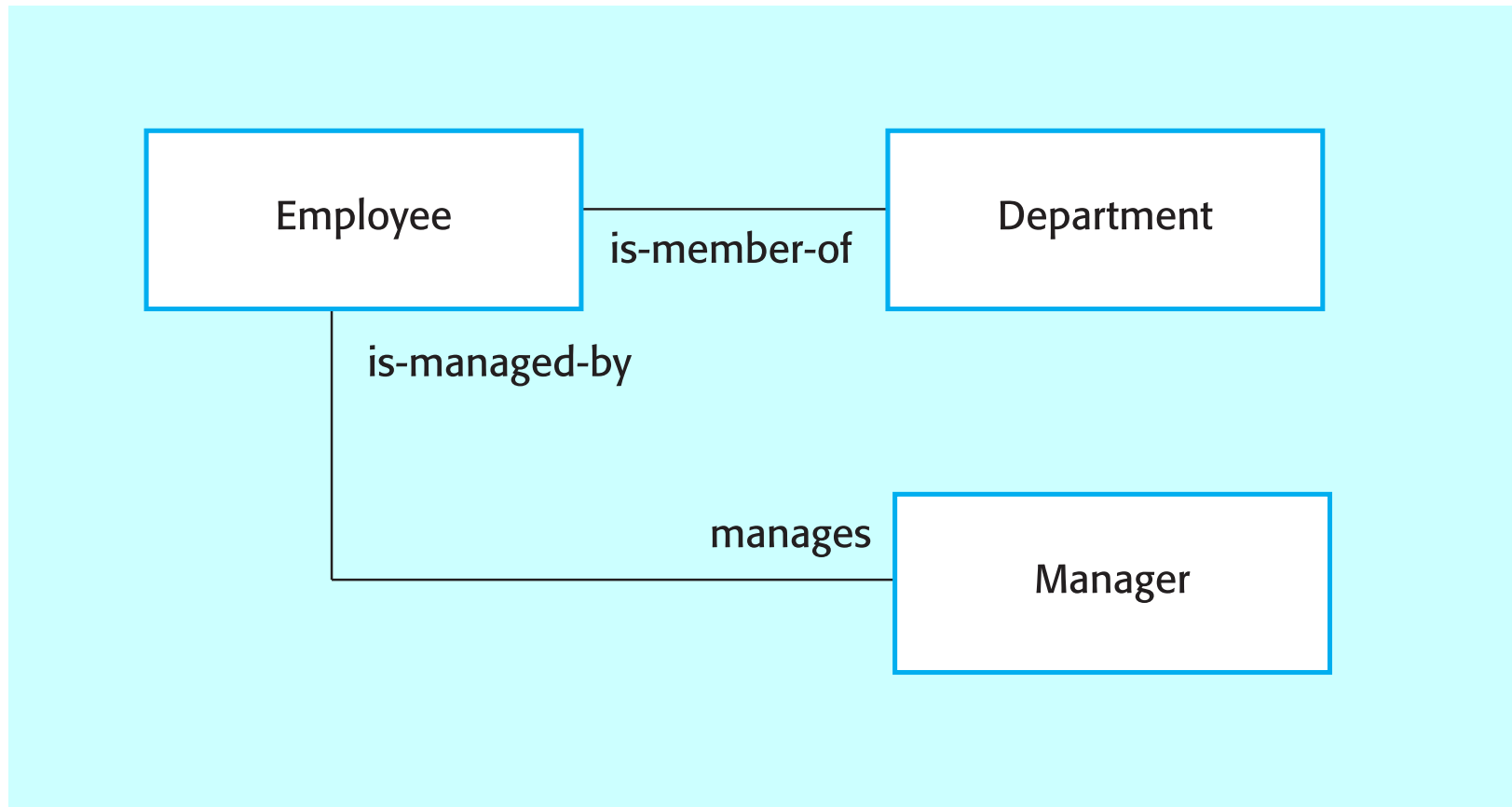


Designing your project the *OO* way

- Design models
- Design process
- **Association model**
- Use case model
- The Class Diagram
- The Collaboration Diagram
- Sequence model



An association model



Designing your project the *OO* way

- Design models
- Design process
- Association model
- **Use case model**
- The Class Diagram
- The Collaboration Diagram
- Sequence model

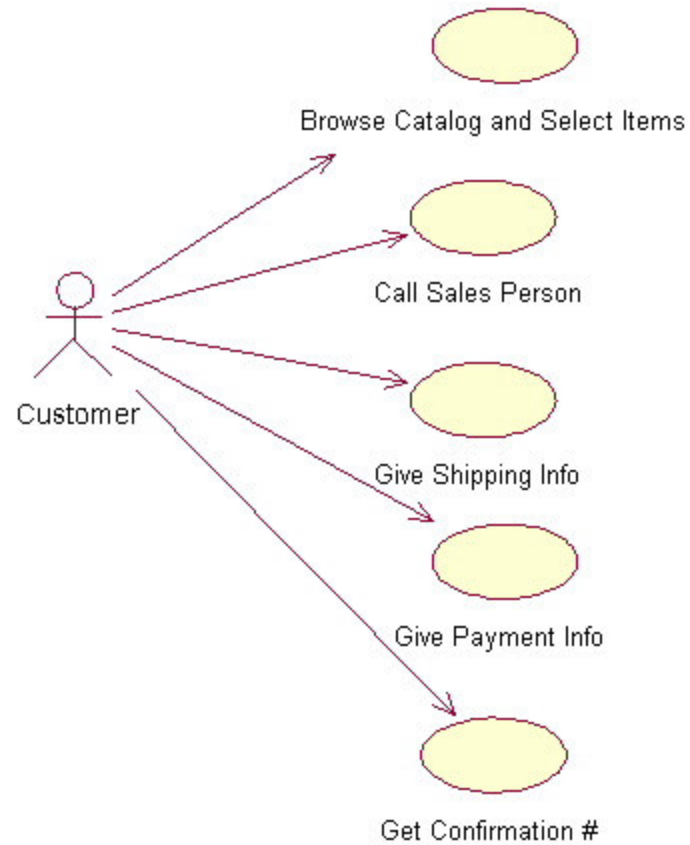


Use-case models

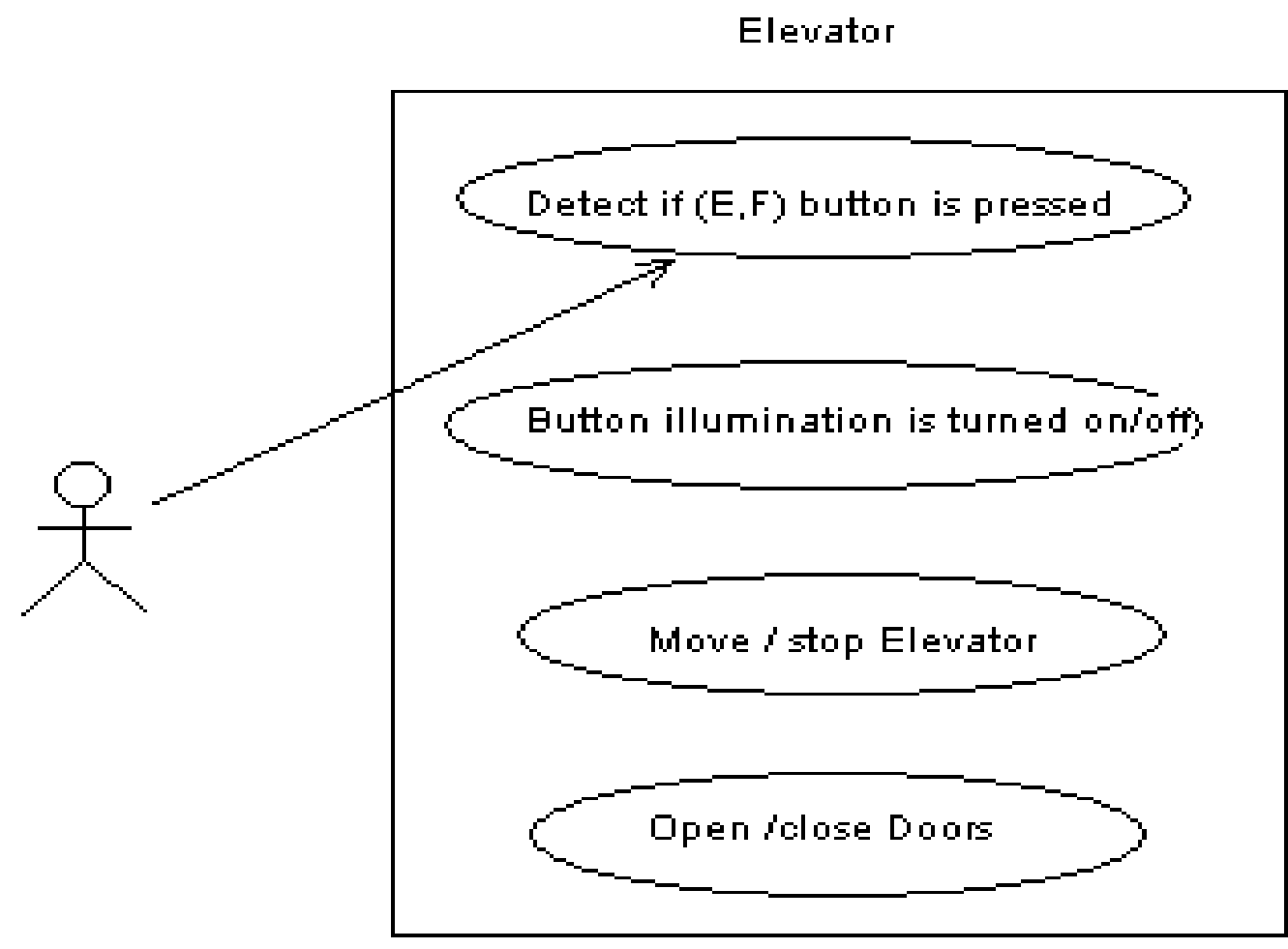
- Use-case models are used to represent each interaction with the system.
- A use-case model shows the system features as ellipses and the interacting entity as a stick figure.
- Made of actors and goals



Use-case: a customer placing an order



Use case: an elevator



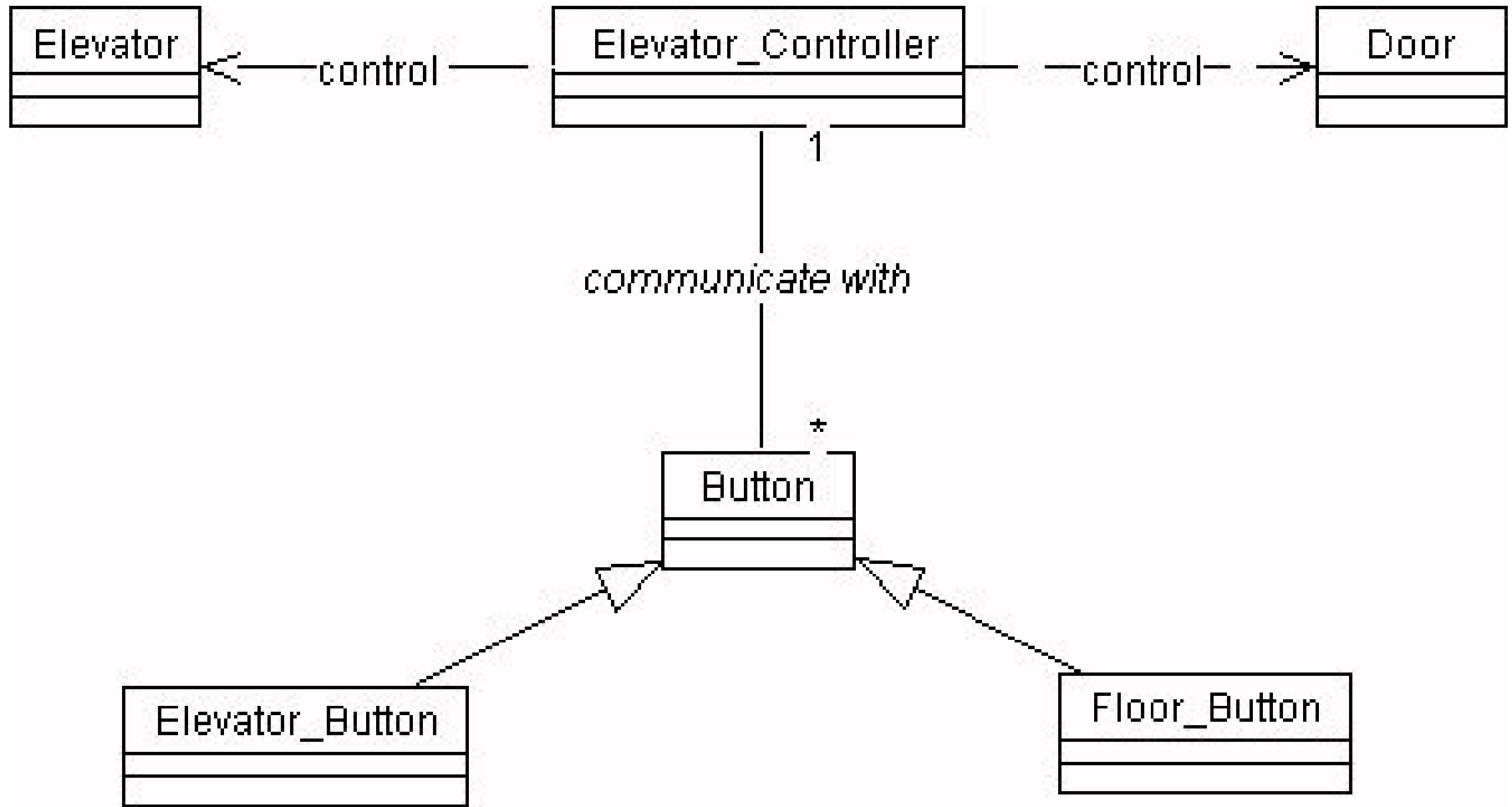
Managing an elevator

Elevator basic scenario that can be extracted from Use Case Diagram:

- * Passenger pressed floor button
- * Elevator system detects floor button pressed
- * Elevator moves to the floor
- * Elevator doors open
- * Passenger gets in and presses elevator button
- * Elevator doors closes
- * Elevator moves to required floor
- * Elevator doors open
- * Passenger gets out
- * Elevator doors closes



Elevator classes



Further objects and object refinement

- Use domain knowledge to identify more objects and operations
 - elevators should have a unique identifier;
 - Buttons outside and inside an elevator operate on different manners.
- Active or passive objects
 - In this case, objects are passive and collect data on request rather than autonomously. This introduces flexibility at the expense of controller processing time.

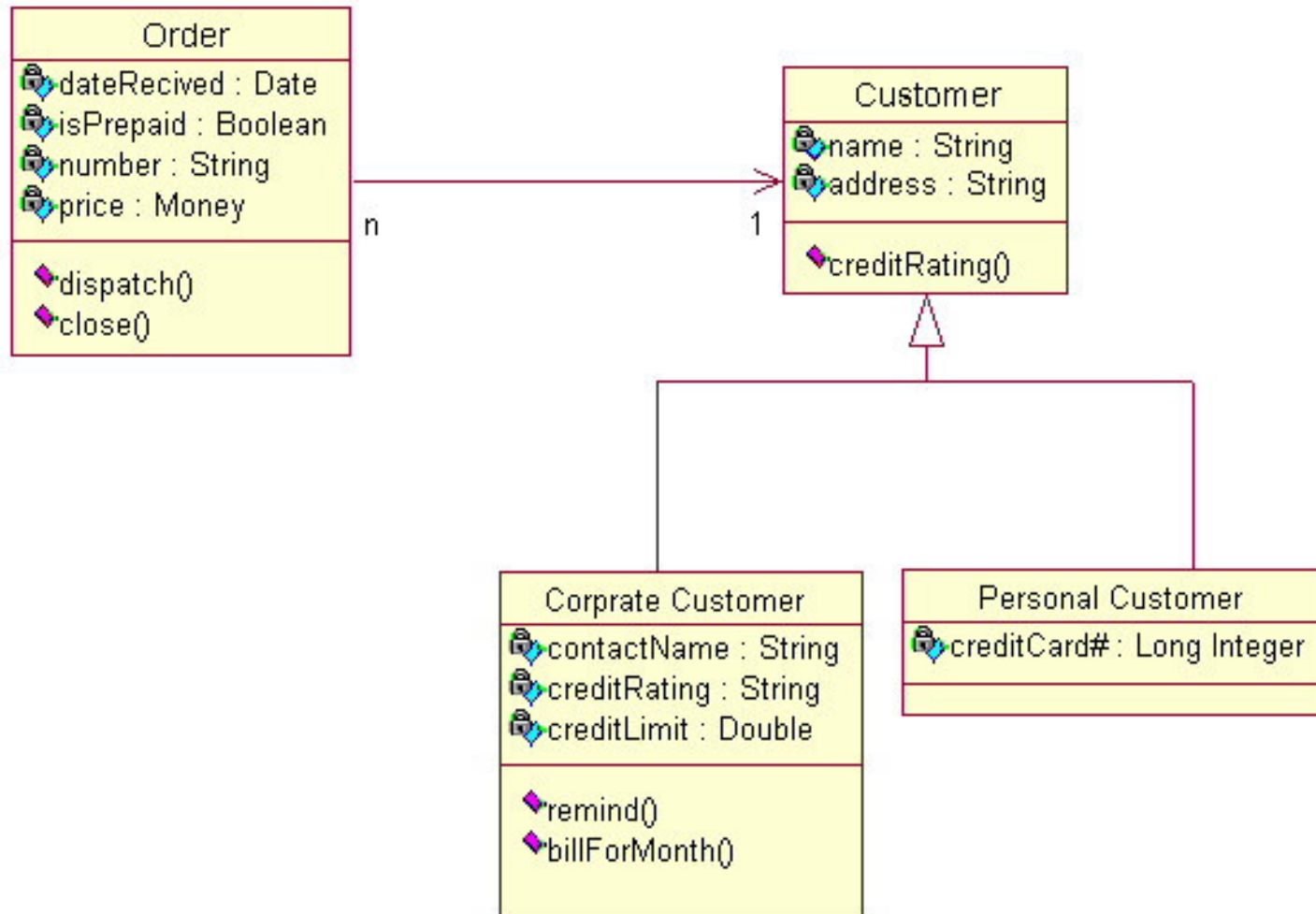


Designing your project the *OO* way

- Design models
- Design process
- Association model
- Use case model
- **The Class Diagram**
- The Collaboration Diagram
- Sequence model



A customer placing orders: class Diagram



Designing your project the *OO* way

- Design models
- Design process
- Association model
- Use case model
- The Class Diagram
- **The Collaboration Diagram**
- Sequence model

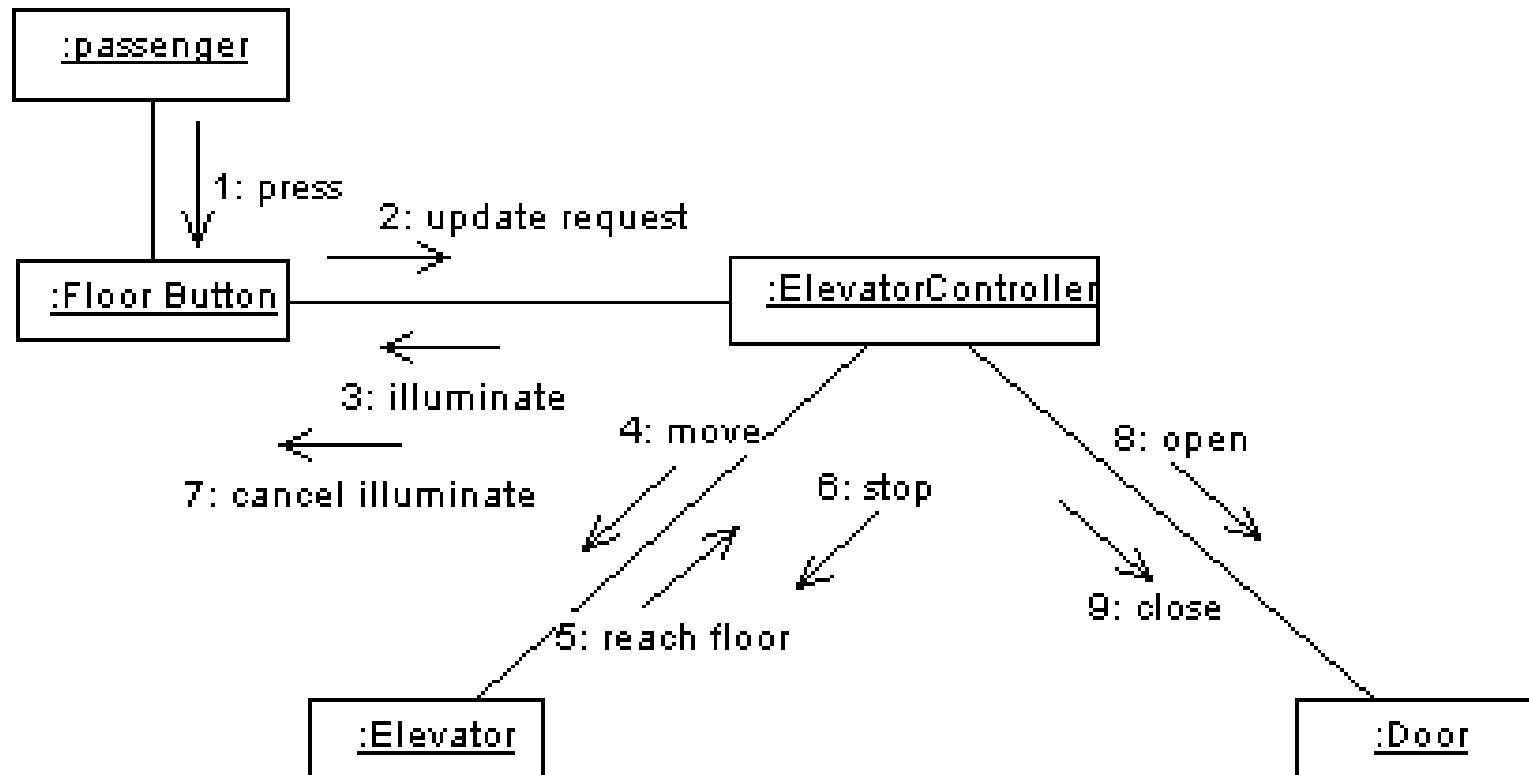


Collaboration diagram

- Describes the set of interactions between classes or types
- Shows the relationships among objects



Elevator collaboration



Designing your project the *OO* way

- Design models
- Design process
- Association model
- Use case model
- The Class Diagram
- The Collaboration Diagram
- Sequence model

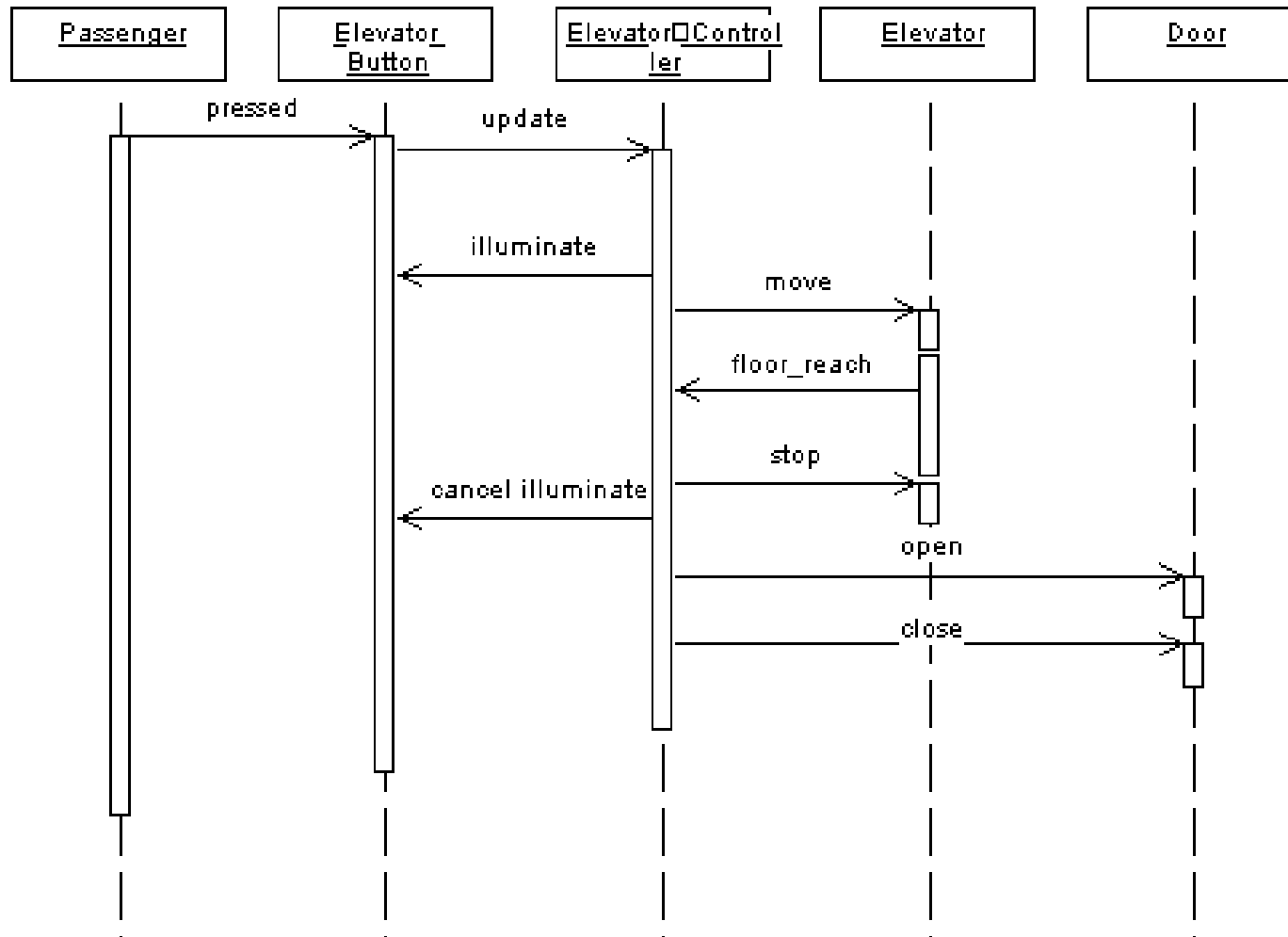


Sequence models

- Sequence models show the sequence of object interactions that take place
 - Objects are arranged horizontally across the top;
 - Time is represented vertically so models are read top to bottom;
 - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
 - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.



inside elevator button sequence



Statecharts

- Show how objects respond to different service requests and the state transitions triggered by these requests
 - If the door is in “closed” state, it will go in “open” state when receiving open() command from the controller

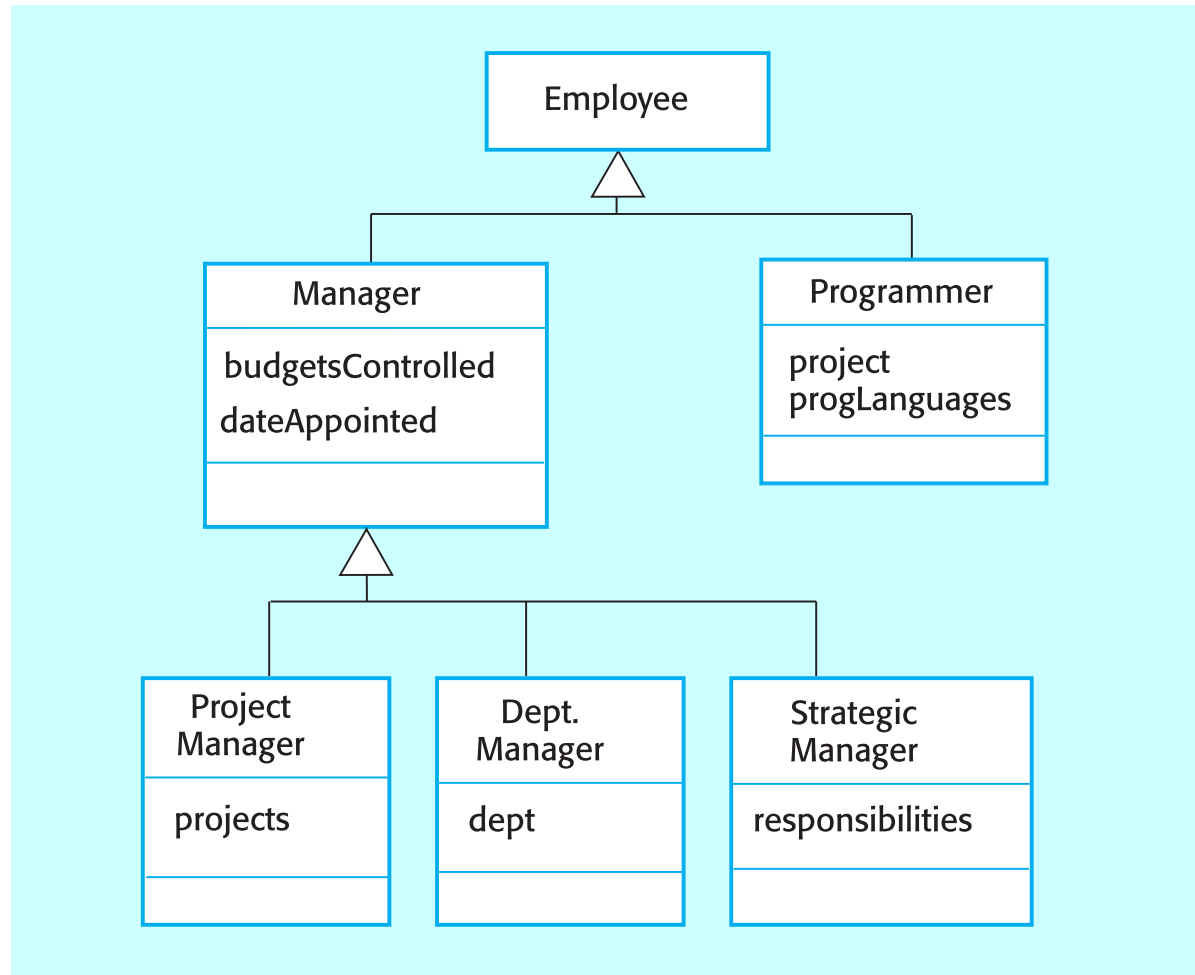


Object interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.



Interface



Changes required

- Add a new kind of elevator, an emergency elevator used in hospitals.
- Add an operation **emergency call**.
- ...



Agenda

- Object-oriented, the hype
- Objects and classes
- Generalisation and inheritance
- Designing your project
- Design process
- **Additional example**
- Object-oriented in SAP
- Summary

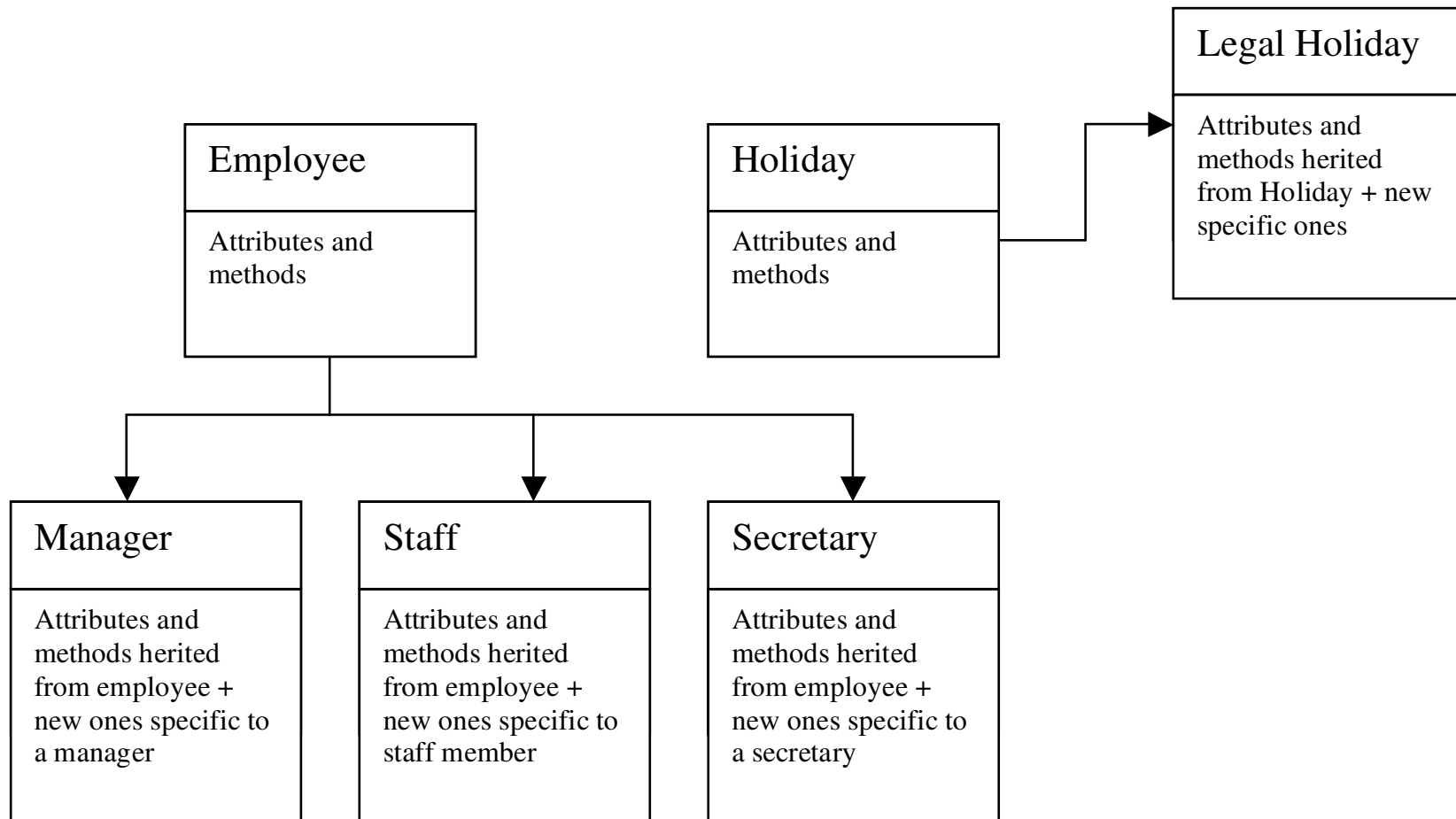


Additional example: employee holiday management

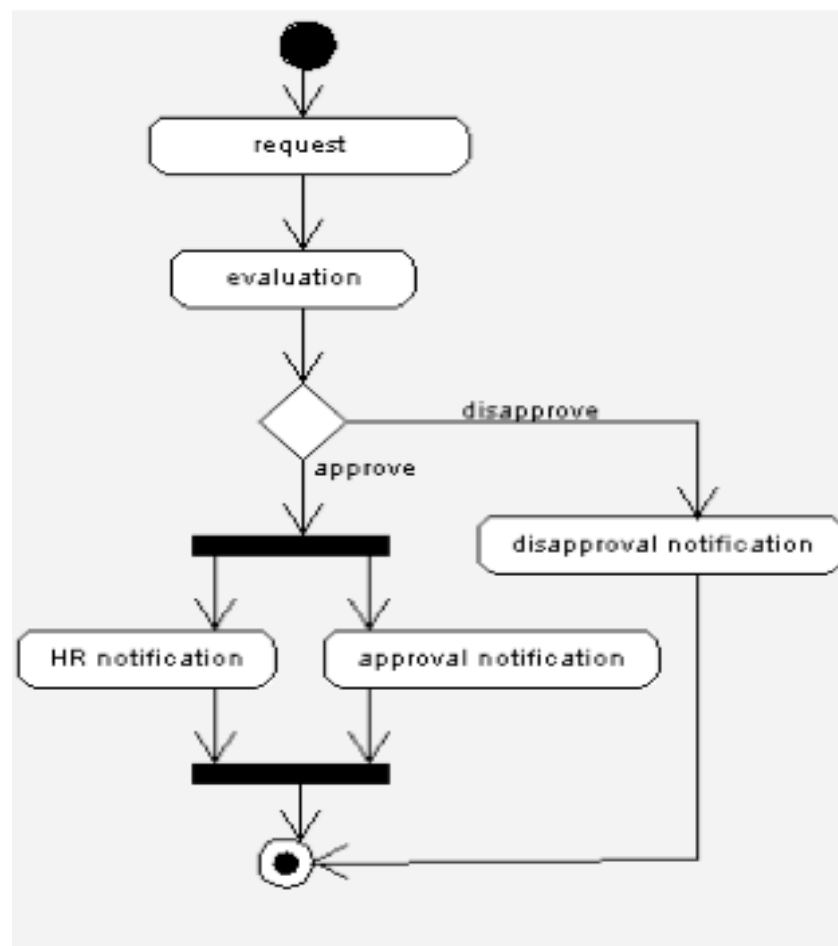
- The company is divided in a simple hierarchy: managers and their staff. It has an administrative cell that handles holiday requests, posted by an employee and approved by his manager.
 - We want to handle legal holiday requests.
- ⇒ First step: open your favorite UML drawing software or get a pencil and a sheet of paper.
- ⇒ Secondly: identify the classes



Classes



UML diagrams: activity diagram

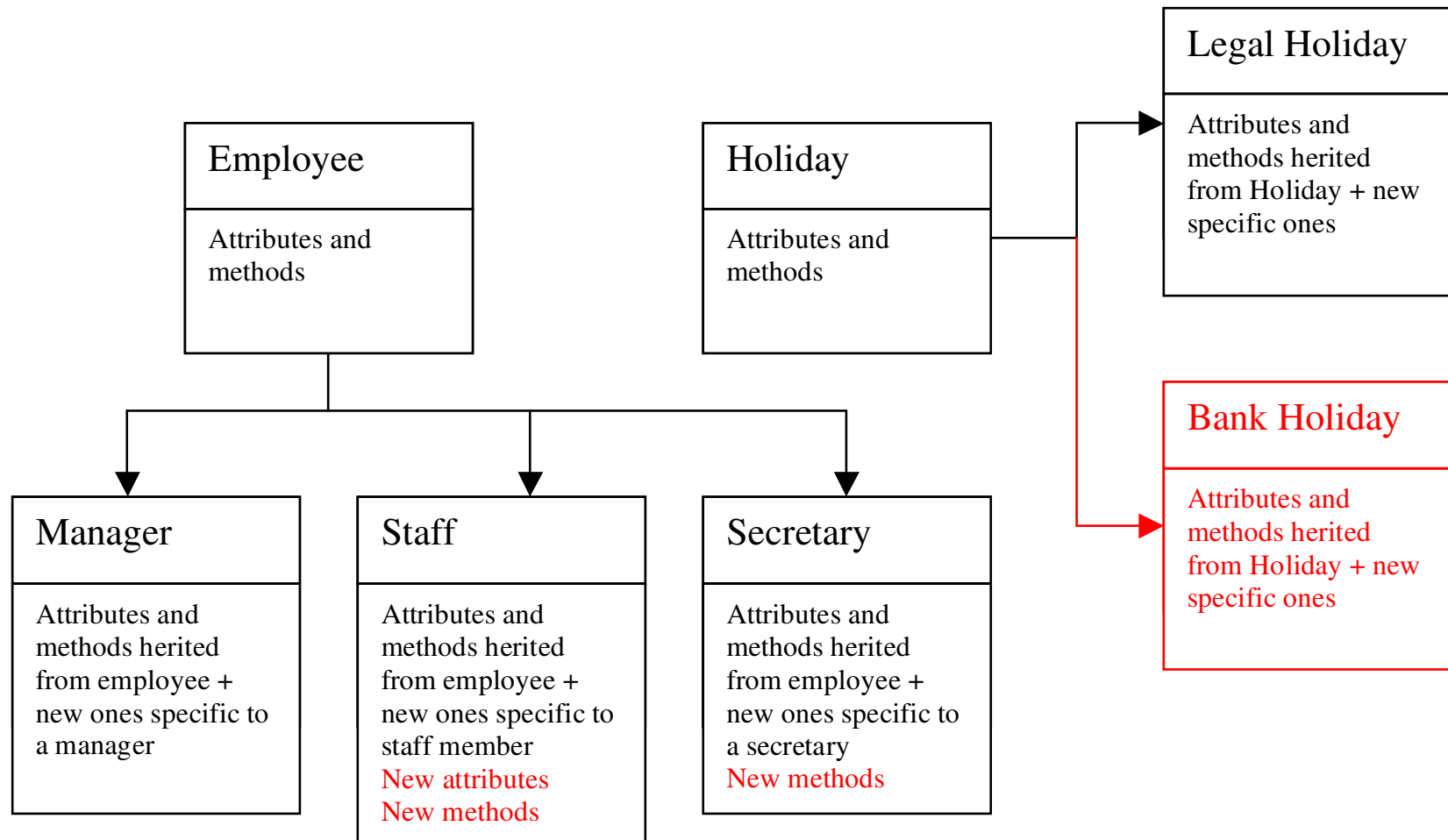


Upgrades and further requests

- Easy to add new employees, new types of holidays
- New processes in the flow are easily added, e.g. once the holiday is processed by the administration there could be specific feedback sent to the Finance division or interactions in the planning calendar of the manager
- No changes to the existing process
- Only extensions
- Flexibility, coherence, readability, efficiency.



Upgrades and further requests



Agenda

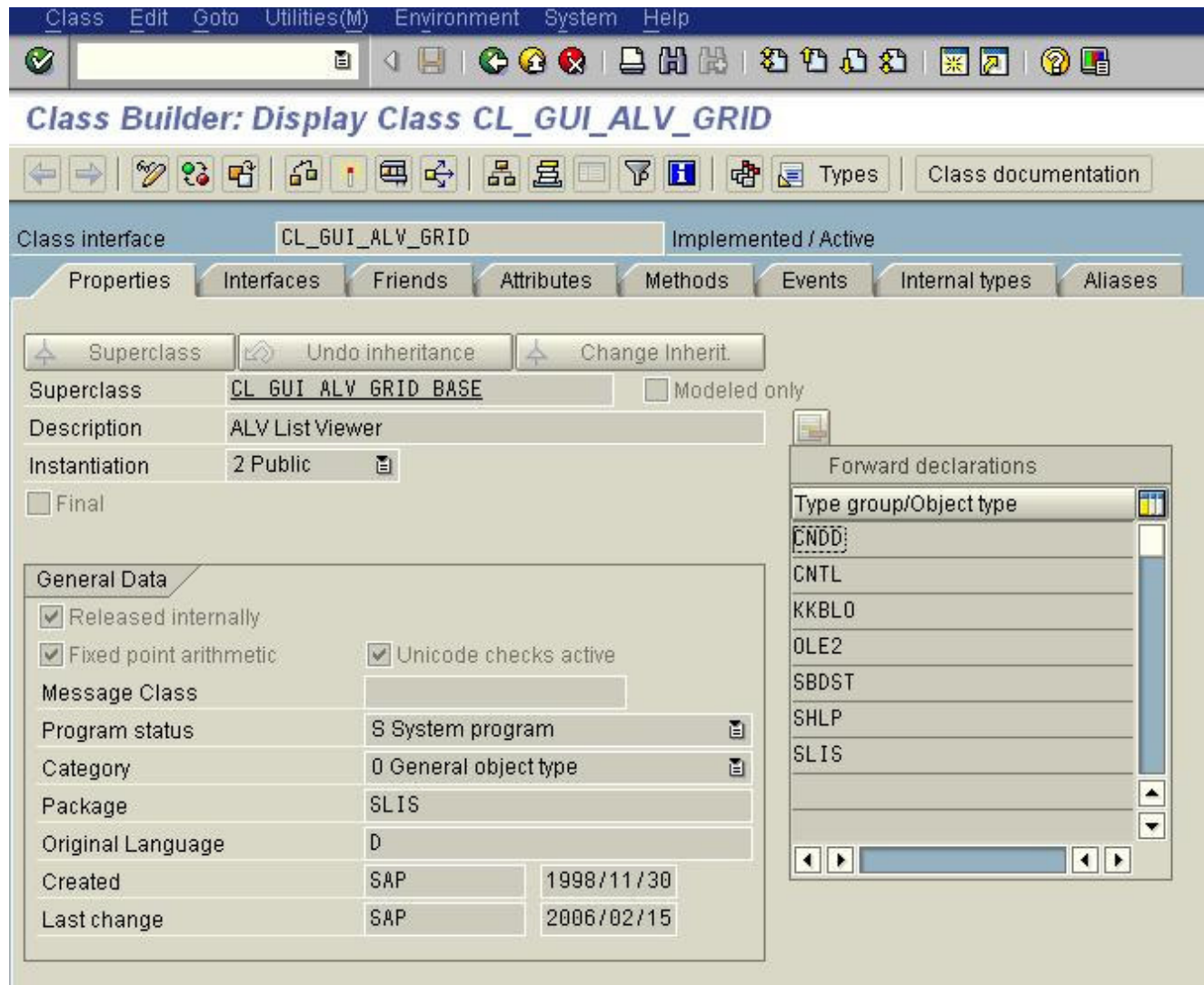
- Object-oriented, the hype
- Objects and classes
- Generalisation and inheritance
- Designing your project
- Additional example
- **Object-oriented in SAP**
- Summary




Transaction se24



Inheritance



Attributes

Properties Interfaces Friends Attributes Methods Events Internal types Aliases							
 <input checked="" type="checkbox"/> Filter							
Attribute	Level	Visibility	Mo...	Re...	Typing	Associated Type	Description
MT_DETAIL	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_DETM	Detail Output
MT_EXCLUDING_TOOLBAR	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	UI_FUNCTIONS	Excluded ALV Standard To
MT_IDPO	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_IDPO	Convert ID to Position
MT_INFO	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_INFO	Column Information
MT_MENUENTRIES	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_TBME	Toolbar Menu Options
MT_MENUENTRIES_SET	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_TBME	Local Copy
MT_OUTTAB	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type Ref	DATA	Original Data Table
MT_POID	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_POID	Convert Position to ID
MT_REPREP_FCCLS	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_FCCL	ReRe Function Code Clas
MT_SPECIAL_GROUPS	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	LVC_T_SGRP	ALV Control: Table of Field
MT_TOOLBAR	Instance Attribut	Protected	<input type="checkbox"/>	<input type="checkbox"/>	Type	TTB_BUTTON	Toolbar

Methods

Class interface **CL_GUI_ALV_GRID** Implemented / Active

Properties Interfaces Friends Attributes **Methods** Events Internal types Aliases

Parameters Exceptions Filter

Methods	Level	Visi...	Mo...	M...	Description
<u>GET_CURRENT_CELL</u>	Instanc	Publi	<input type="checkbox"/>		Current Row
GET_FILTERED_ENTRIES	Instanc	Publi	<input type="checkbox"/>		Hash Table of Filtered Entries
GET_FILTER_CRITERIA	Instanc	Publi	<input type="checkbox"/>		Get Filter Criteria
GET_FRONTEND_FIELDCATALOG	Instanc	Publi	<input type="checkbox"/>		Get Current Field Catalog from Frontend
GET_FRONTEND_LAYOUT	Instanc	Publi	<input type="checkbox"/>		Get Current Layout from Frontend
GET_FRONTEND_PRINT	Instanc	Publi	<input type="checkbox"/>		Get Current Print from Frontend
GET_SCROLL_INFO_VIA_ID	Instanc	Publi	<input type="checkbox"/>		Use ID to Get Scroll Information
GET_SELECTED_CELLS	Instanc	Publi	<input type="checkbox"/>		Get Selected Cells
GET_SELECTED_CELLS_ID	Instanc	Publi	<input type="checkbox"/>		Get Selected Cell IDs
GET_SELECTED_COLUMNS	Instanc	Publi	<input type="checkbox"/>		Get Selected Columns
GET_SELECTED_ROWS	Instanc	Publi	<input type="checkbox"/>		Get Selected Rows
GET_SORT_CRITERIA	Instanc	Publi	<input type="checkbox"/>		Get Sort Criteria
GET_SUBTOTALS	Instanc	Publi	<input type="checkbox"/>		Get Subtotals Table

Agenda

- Object-oriented, the hype
- Objects and classes
- Generalisation and inheritance
- Designing your project
- Additional example
- Object-oriented in SAP
- Summary



Key points

- OOD is an approach to design so that design components have their own private state and operations.
- Objects should have constructor and inspection operations. They provide services to other objects.
- Objects may be implemented sequentially or concurrently.
- The Unified Modeling Language provides different notations for defining different object models.



Key points

- A range of different models may be produced during an object-oriented design process. These include static and dynamic system models.
- Object interfaces should be defined precisely
- Object-oriented design potentially simplifies system evolution.

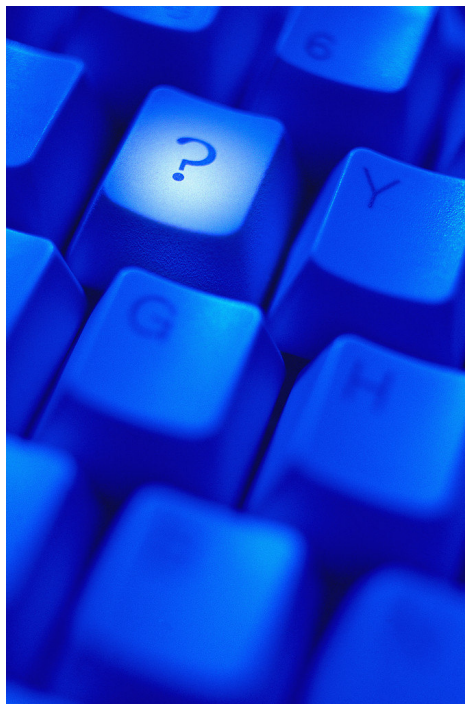


Conclusion

- The building of an OO project reduces development time while increasing the importance of analysis.
- Will reduce maintenance and upgrade time
- OO design reduces further development effort
- Needs a good analytical basis and design
- Current software evolution trend



Questions?



usg innotiv
professionals in ict
onvoorwaardelijk betrouwbaar