Advanced Analysis

UML - More than a graphical notation

Qualified Associations and Association classes

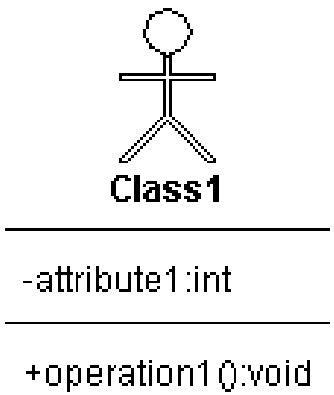
Why we need more than diagrams?

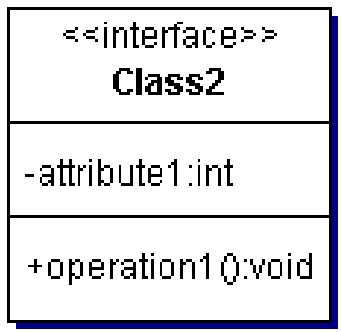
- Graphical notations
  - Advantage: good for displaying structural aspects of the system
  - Disadvantage: can quickly become very complex
- Solution: supplement graphical notations with textual annotations

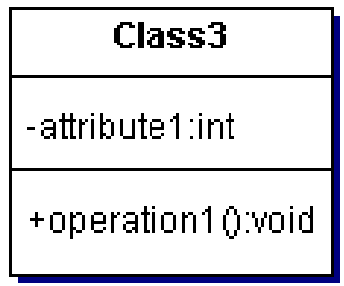
Extending UML (1)

- Stereotypes
  - Introduce a variation on modelling element semantics
  - `<<name>>` or iconic
  - Some stereotypes are pre-defined in UML
  - Profile: a purposeful set of stereotypes to address a design modelling issue

Extending UML (2)







"Icon" stereotype

"label" stereotype

No stereotype

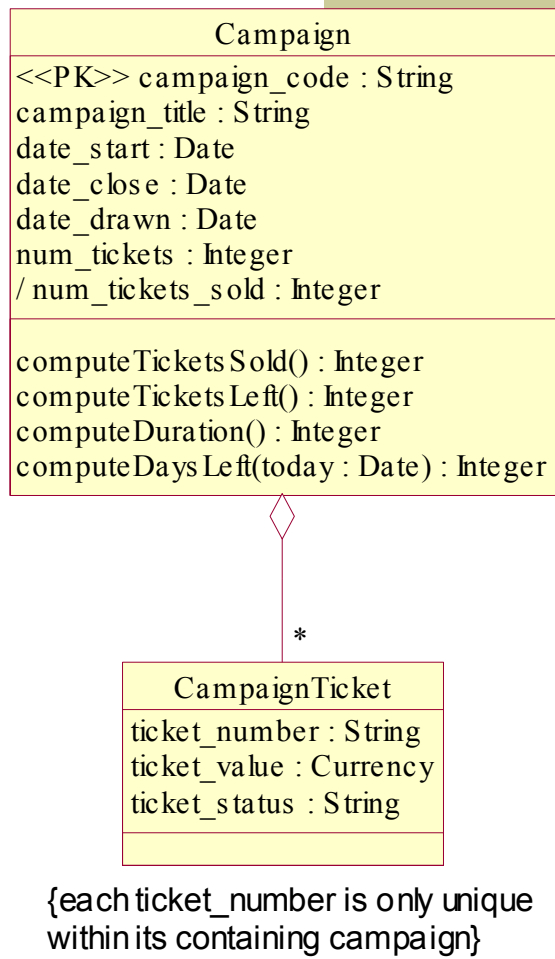
Where else have you seen stereotypes?

More Expressive Diagrams (1)

- Constraints to modelling elements
  - {text}
  - It is important to keep a balance on the amount of textual information attached to diagrams
- Constraints and Stereotypes
  - Different, but some semantic variations may be constraints

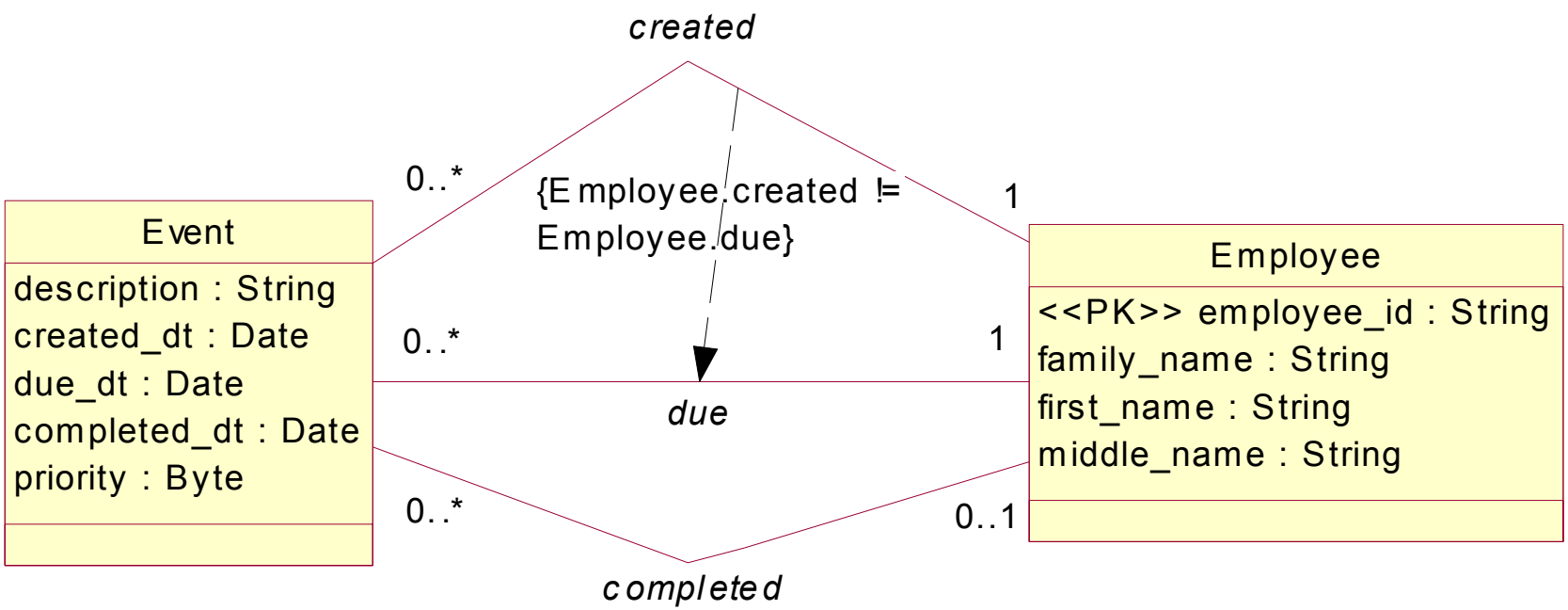
More Expressive Diagrams (2)

- Telemarketing
  - All tickets are numbered. The numbers are unique across all tickets in a campaign



More Expressive Diagrams (3)

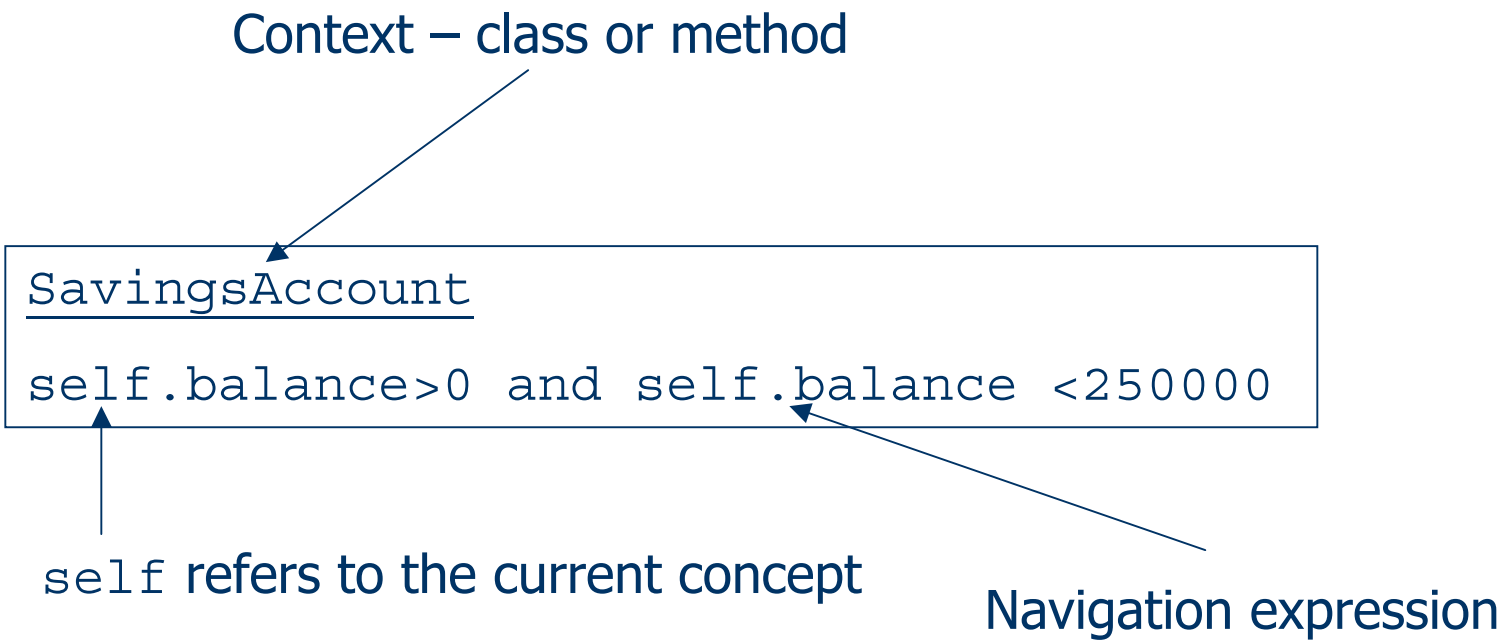
- Contact Management
  - The system does not handle scheduling of events by employees to themselves. This means that an employee who created an event must not be the same as the employee who is due to perform that event.



More Expressive Diagrams (4)

- Object Constraint Language (OCL)
  - OCL expression elements
    - Context – on which modelling elements the constraint applies
    - Navigation expressions – referring to other modelling elements that may be relevant to the constraint being defined
    - Assertions – assertions about the relationships between the referred modelling elements

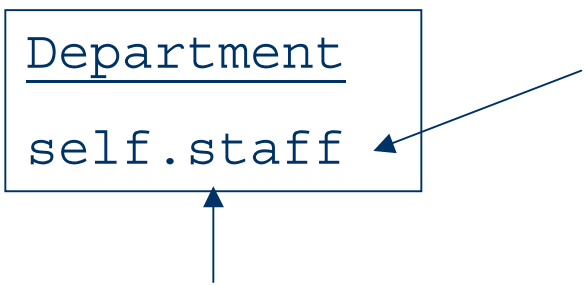
More Expressive Diagrams (5)



balance is an attribute of class SavingsAccount

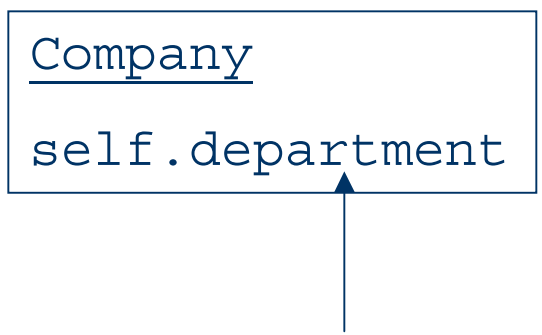
More Expressive Diagrams (6)

- Navigation expressions
  - Links
    - Keep in mind that ambiguity is not acceptable



Role name at the opposite side of the association

The set of objects currently linked to the object through the specified link – the employees currently working for the department



Instead of a role name the name of the class on the other side

More Expressive Diagrams (7)

- Collections
  - Multiplicity constraints denote the number of retrieved objects

- Iterated traversal

More Expressive Diagrams (8)

- Object and Collections
  - Operations on objects
    - Types in OCL: basic types (boolean, integer, real, string) and model types (classes in the UML model)
    - Model types have attributes and operations

```
Person
self.age()
self.contract.grade.salary
```

More Expressive Diagrams (9)

- Different types of collection
  - Set: a single occurrence of each object
  - Bag: multiple occurrences of the same object
    - When more than one association with multiplicity greater than one is accessed then the returned collection is a bag

```
Department
self.staff.contract.grade
```

More Expressive Diagrams (10)

- Operations on collections

```
Department
staff.contract.grade.salary->sum()
```

```
Department
staff.contract.grade->asSet()->size
```

```
Company
employee->select(p:Person | p.contract.grade.salary>50000)
```

```
Company
employee->select(contract.grade.salary>50000).manager
```

```
Department
staff->collect(p:Person | p.age())
```

More Expressive Diagrams (11)

- OCL Constraints

```
Person
self.employer=self.department.company
```

```
Company
employee->select(age()<18)->isEmpty
employee->select(age()<18)->size=0
```

```
Person
employer.grade->include(contract.grade)
```

```
Department
company.employee->includesAll(staff)
```

More Expressive Diagrams (12)

Person
```
self.age()>50 implies self.contract.grade.salary>25000
```

Company
```
self.grade->forAll(g | not g.contract->isEmpty())
```

Department
```
staff->exists(e | e.manger->isEmpty())
```

Grade
```
Grade.allInstances->forAll(g | g.salary > 20000)
salary > 20000
```

Grade
```
Grade.allInstances->forAll(g: Grade |
      g <> self implies g.salary <> self.salary
```

More Expressive Diagrams (13)

- Stereotyped OCL constraints
  - Class invariants
    - Properties of the class that are intended to be at all times for all instances of the class
      - Usually for constraints that restrict possible values of attributes
  - Preconditions and postconditions
    - Precondition: something that must be true just before an operation is called
    - Postcondition: something that must be true just after the operation has completed

More Expressive Diagrams (14)

- A precondition is usually expressed as a constraint relating the attributes of a class instance and the actual parameters of the operation being specified
- A postcondition typically specify the effect of an operation by comparing the attribute values before and after the execution of the operation

```
SavingsAccount
balance > 0 and balance < 250000
```

```
SavingsAccount::withdraw(amt)
pre: amt < balance
post: balance = balance@pre - amt
```

More Expressive Diagrams (15)

- Design by contract
  - B. Meyer – Eiffel language
    - Caller guarantees precondition
    - Callee guarantees postcondition
    - If both sides of the contract are satisfied then invariant holds
  - Stresses class encapsulation
  - Defensive programming: always check the preconditions

More Expressive Diagrams (16)

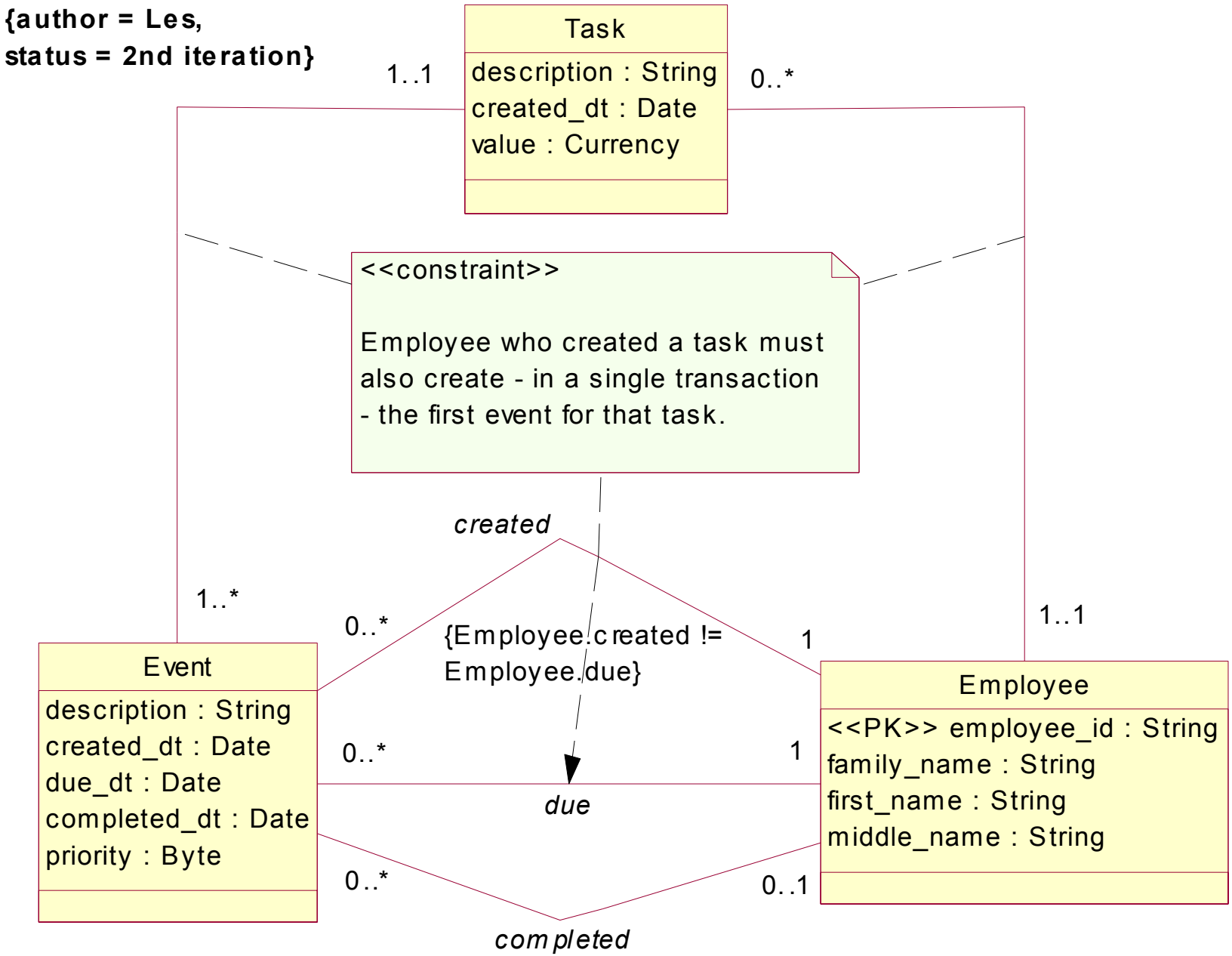
- Aspects of a contract
  - The intent or purpose of the operation
  - The operation signature including return type
  - An appropriate description of the logic
  - Other operation called, whether in the same object or other objects
  - Events transmitted to other objects
  - Attributes set during the operation's execution
  - The response to exceptions
  - Any non-functional requirements

More Expressive Diagrams (17)

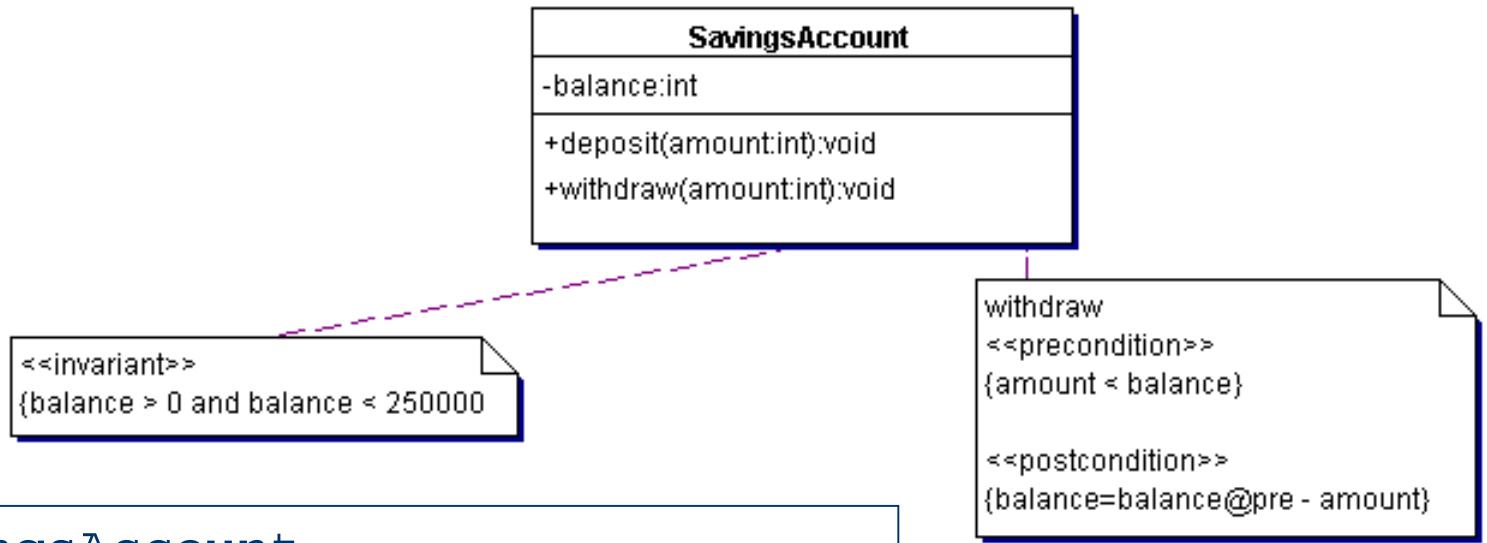
- Notes and Tags
  - Notes: any kind of comment or constraint attached to a modelling element
  - Tags: name value pairs of arbitrary information or constraints
    - `{tag = value}`
    - A typical use of tags is in providing project management information

More Expressive Diagrams (18)

- Contact Management
  - An employee who created a task must also create – in the same transaction – the first event for that task
  - Show on the diagram that the analyst is Les and the diagram is in its second iteration


{author = Les,
status = 2nd iteration}
Task
description : String
created_dt : Date
value : Currency
1..1
0..*
<<constraint>>
Employee who created a task must also create - in a single transaction - the first event for that task.
created
1..*
0..*
{Employee.created != Employee.due}
1
1..1
Event
description : String
created_dt : Date
due_dt : Date
completed_dt : Date
priority : Byte
Employee
<<PK>> employee_id : String
family_name : String
first_name : String
middle_name : String
0..*
1
due
0..*
0..1
completed

More Expressive Diagrams (20)




```
SavingsAccount
balance > 0 and balance < 250000
```

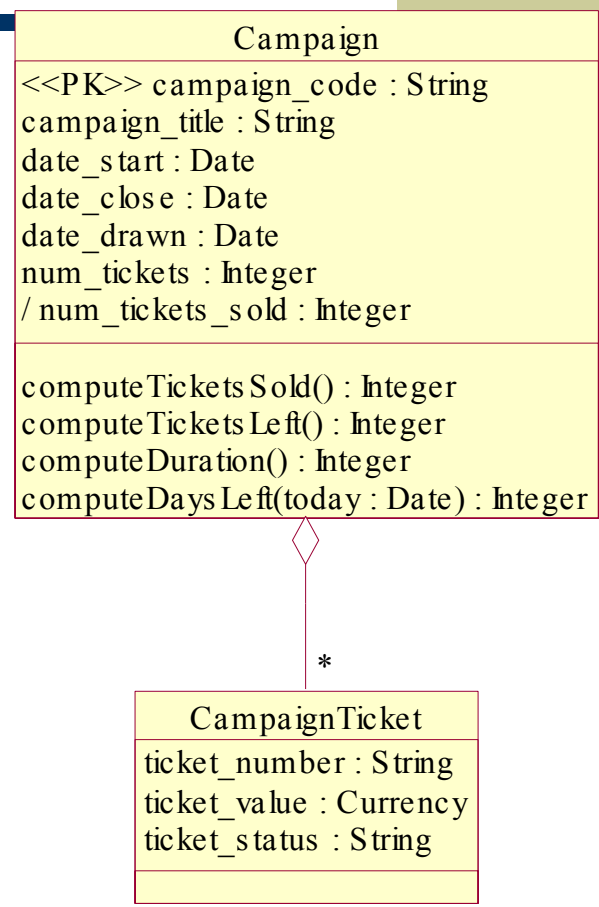
```
SavingsAccount::withdraw(amt)
pre: amt < balance
post: balance = balance@pre - amt
```

Derived Information (1)

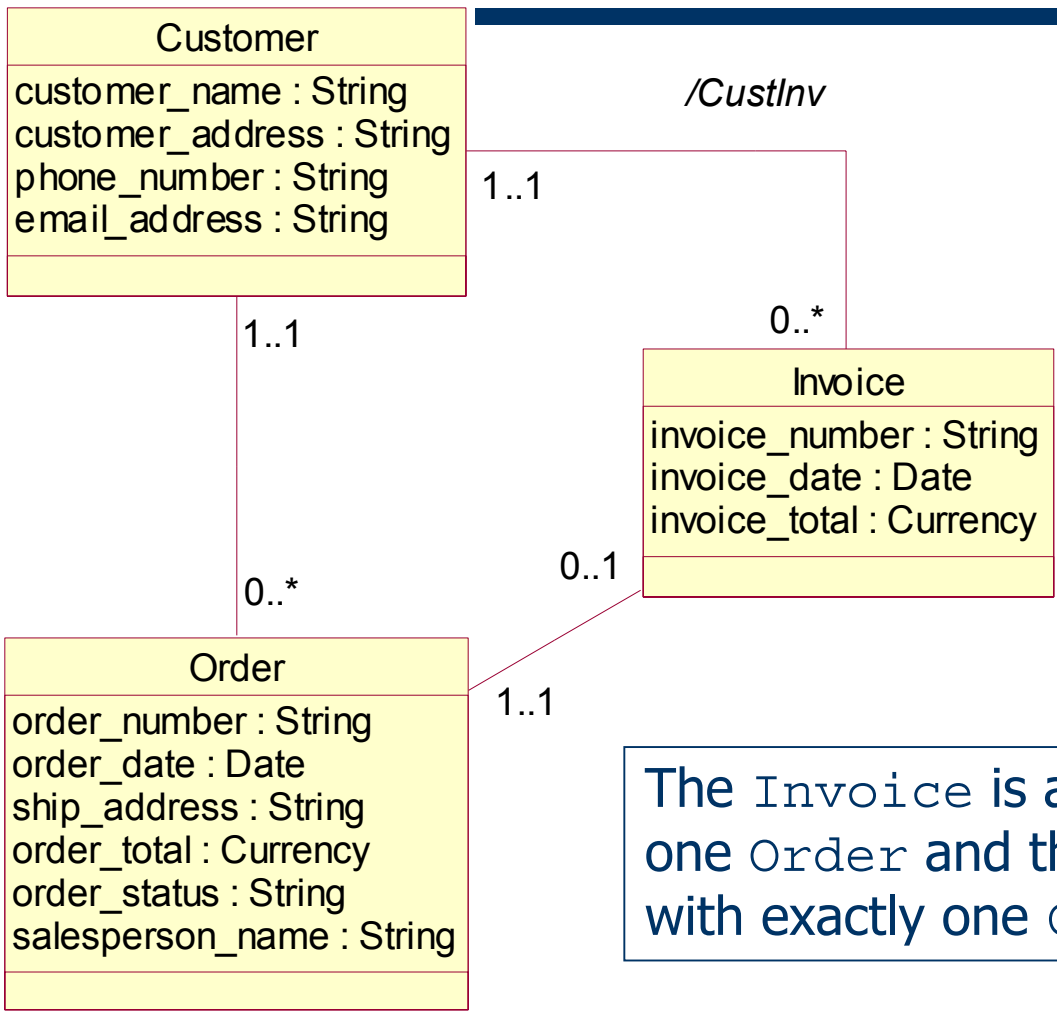
- Derived information can be computed from other model elements! – constraint
  - Applies on attributes and associations
  - In design may indicate optimisations
  - (/) in front of the derived attribute or association name

Derived Information (2)

- Telemarketing
  - The number of sold tickets computed by the operation
- `computeTicketsSold()`
  - Follows the aggregation links, checks the status of each ticket and if it is sold adds it to the count

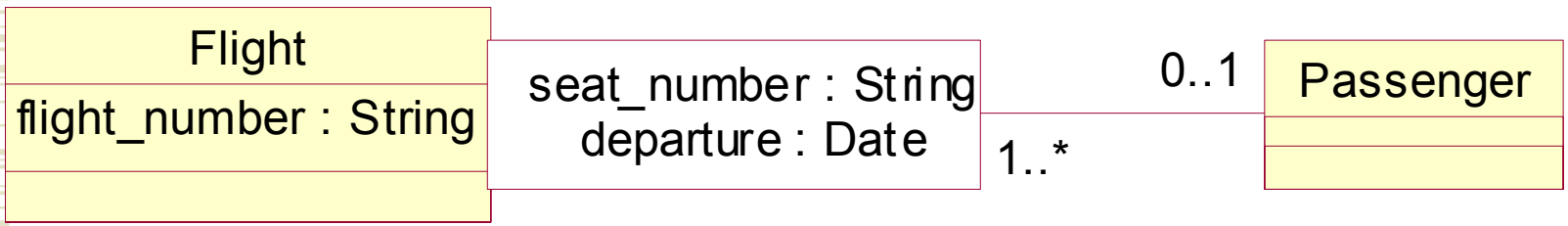


Derived Information (2)



The `Invoice` is associated with exactly one `Order` and the `Order` is associated with exactly one `Customer`

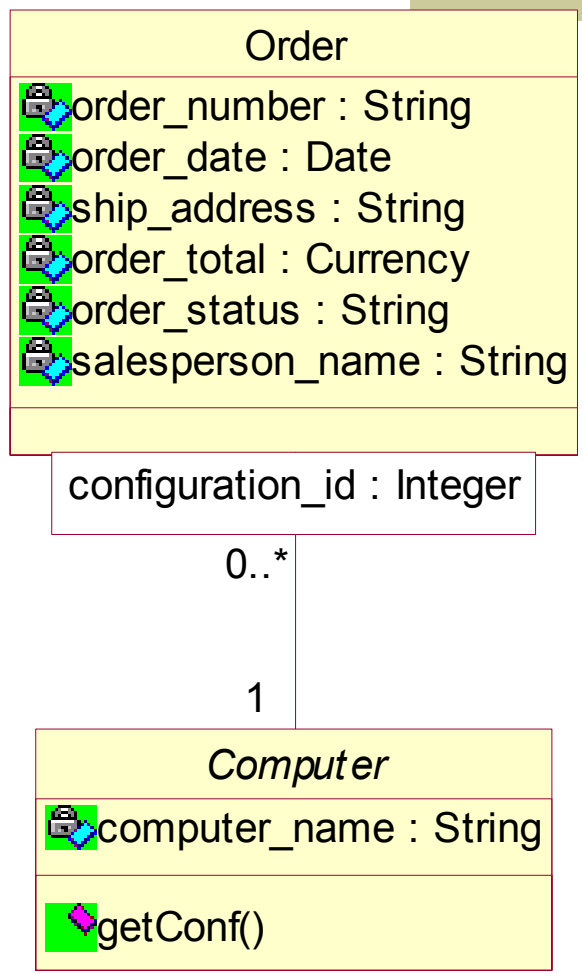
Qualified Associations (1)



- An association with an attributed compartment (qualifier)
- Attributes serve as an index key for traversing the association
  - Key: qualified object + qualifier value
  - Traversal (forward or reverse) and multiplicities

Qualified Associations (2)

- A single order item per computer on order
- A computer is ordered after being configured by the client
- `(order_number + configuration_id)` links to a single computer



Qualified Associations (3)

- OCL
  - Traversing qualified associations

```
Person
self.employee
```

```
Company
self.employee[314159].manager
```

  - Using association classes

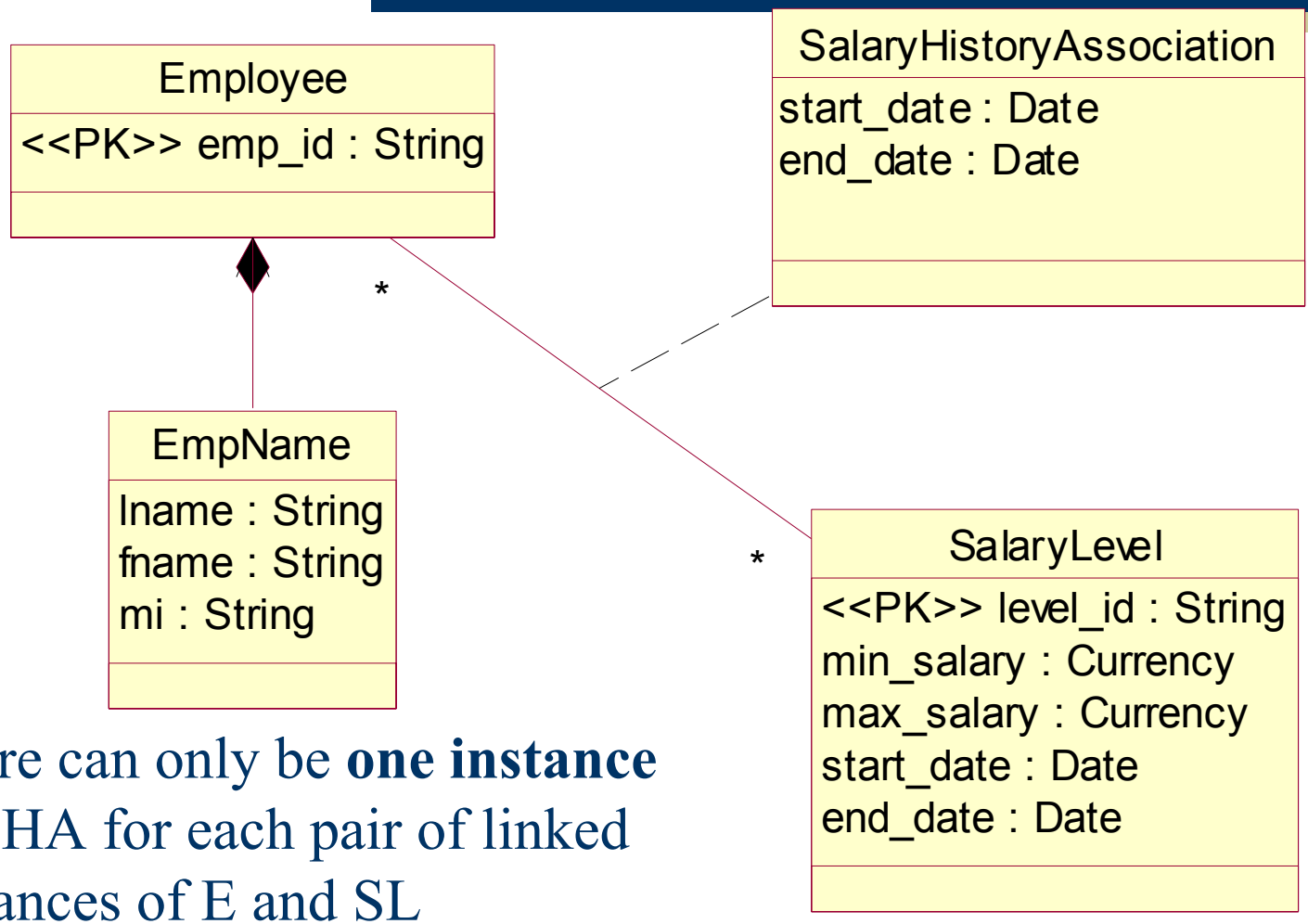
```
Grade
self.contract.employee
```

```
Person
self.contract.grade
```

Association Classes (1)

- Employee database
  - Each employee in an organisation is assigned a unique emp_id. The name of the employee is maintained and consists of the last name, first name and middle initial. Each employee is employed at a certain salary level. There is a salary range for each level, i.e. the minimum and maximum salary. The salary ranges for a level never change. If there is a need to change the minimum or maximum salary, a new salary level is created. The start and end dates, for each salary level, are also kept.

Association Classes (2)

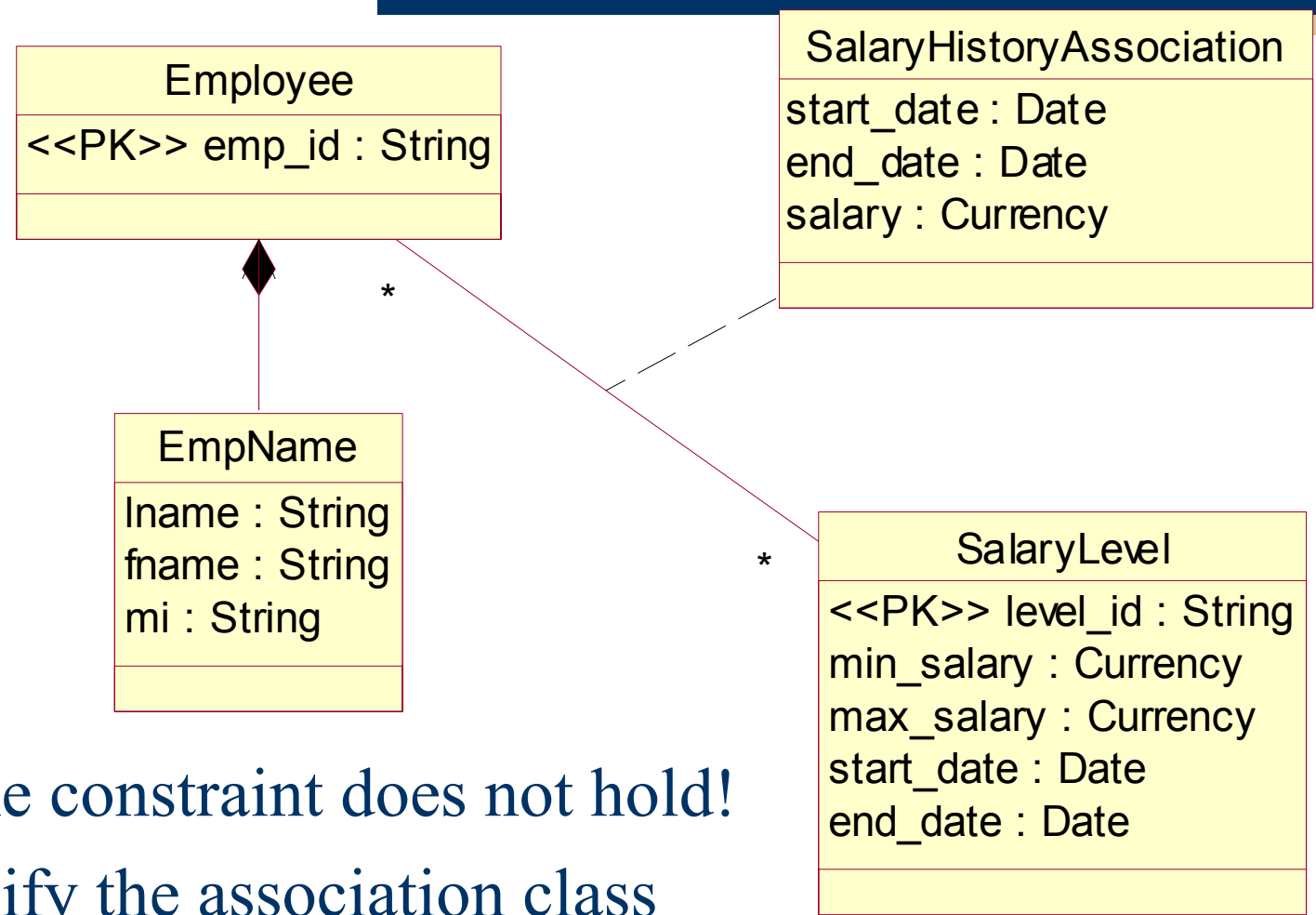


There can only be **one instance** of SHA for each pair of linked instances of E and SL

Association Classes (3)

- Employee database – additional requirement
  - Previous salaries of each employee are kept, including start date and finish date at each level. Any changes of the employee's salary within the same level are also recorded.

Association Classes (4)



The constraint does not hold!

Reify the association class

Association Classes (5)