

MACIASZEK, L.A. (2005):
Requirements Analysis and System Design, 2nd ed.
Addison Wesley, Harlow England, 504p.
ISBN 0 321 20464 6

Chapter 5
Moving from Analysis to Design

© Pearson Education Limited 2005

Topics

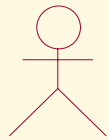
- *Advanced class modeling*
 - *Extension mechanisms*
 - *Visibility, encapsulation and derived information*
 - *Qualified associations and associations as classes*
- *Advanced generalization and inheritance modeling*
 - *Substitutability and inheritance versus encapsulation*
 - *Interface versus implementation inheritance*
- *Advanced aggregation and delegation modeling*
 - *Semantics of aggregation*
 - *Aggregation versus generalization*

Extension mechanisms

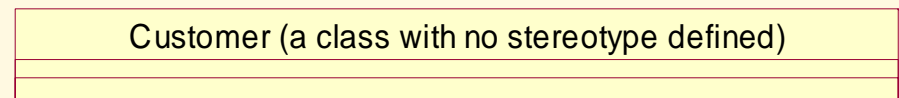
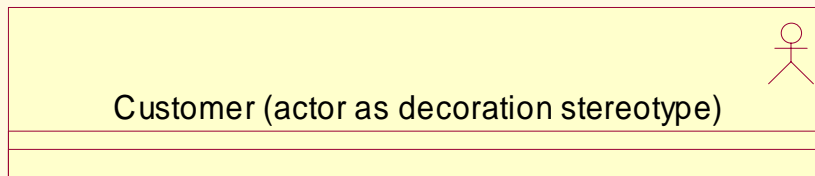
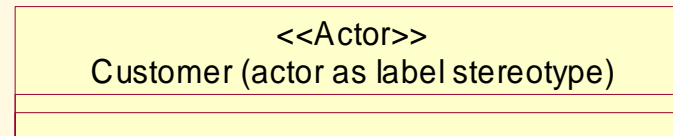
- *specify “how specific UML model elements are customized and extended with new semantics by using*
 - *stereotypes,*
 - *constraints,*
 - *tag definitions, and*
 - *tagged values”*
- *UML profile –coherent set of extensions, defined for specific purposes*

Stereotypes

- *extends an existing UML modeling element*
 - *varies the semantics of an existing element (it is not a new model element per se)*



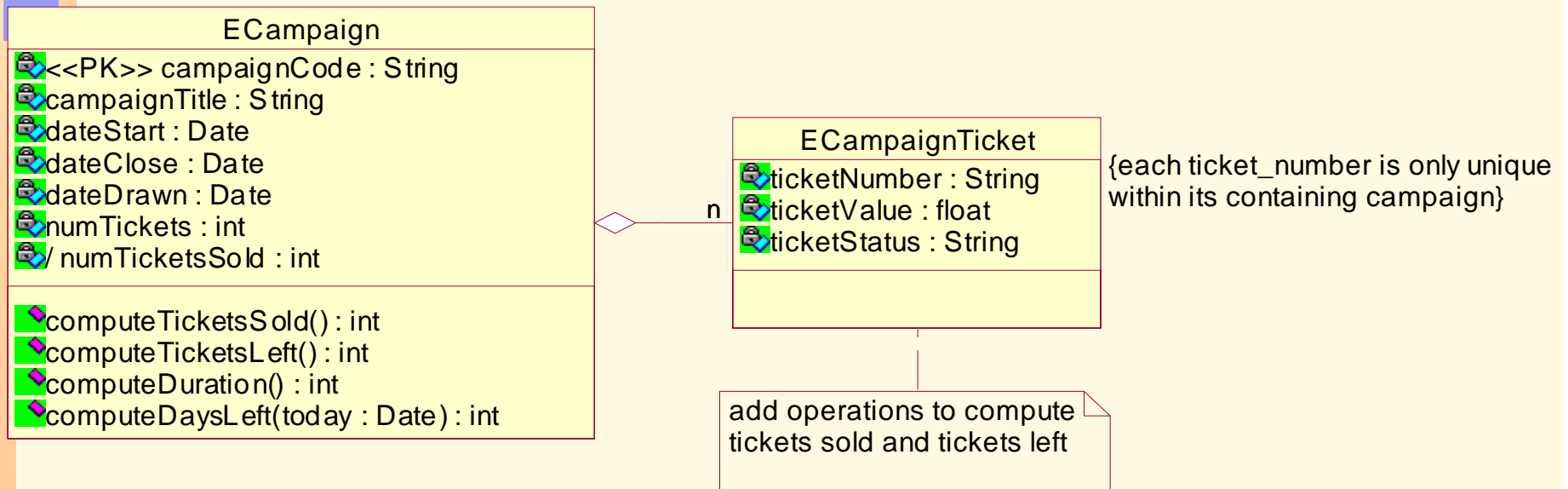
Customer (actor as icon stereotype)



Comments and constraints

- **Comment** - text string attached to a model element
 - can be presented as a UML **note**
- **Constraint** - semantic relationship among model elements that specifies conditions and propositions that must be maintained as true
 - enclosed in braces {...}

to be distinguished from bonus campaign

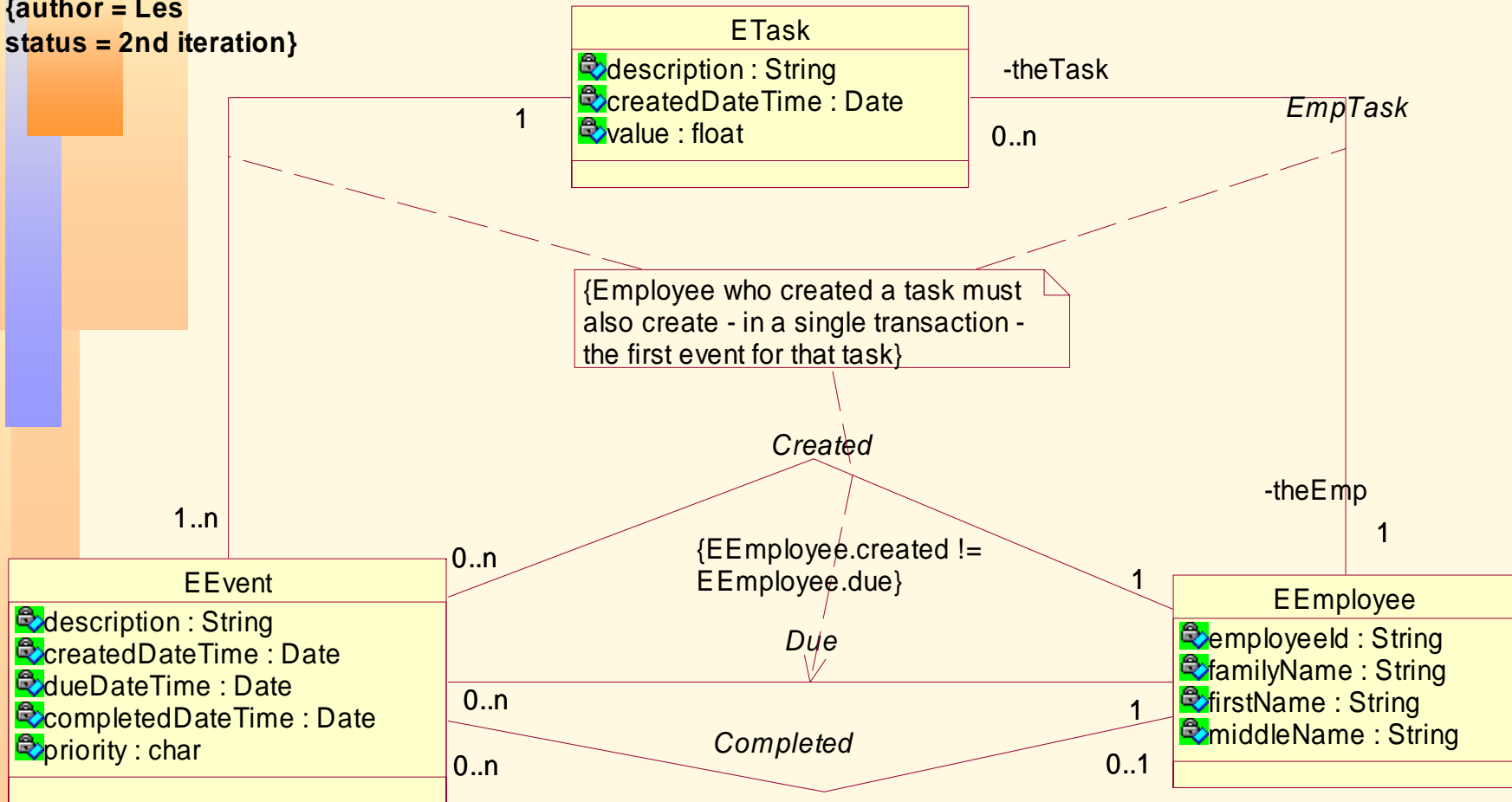


Tags

- **Tagged value** - keyword-value pair that may be attached to any kind of model element
 - the keyword is called a **tag**
 - like **constraints**, tag values represent arbitrary textual information and are written inside curly brackets
 - unlike **stereotypes**, tag values are not meant to extend the UML semantics and create new UML profiles
 - like stereotypes and constraints, few tags are **predefined** in UML
 - typical use of tags is in providing **project management** information









Tag, note, constraint - example

{author = Les
status = 2nd iteration}



Visibility and encapsulation

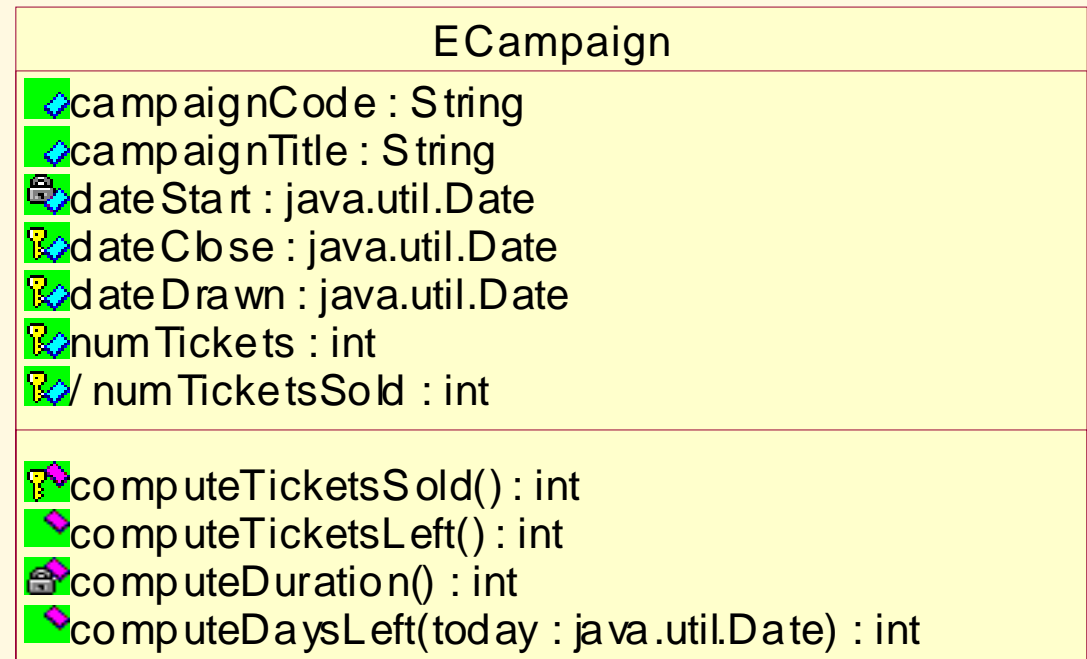
- + for public visibility
- for private visibility
- # for protected visibility
- ~ for package visibility

Visibility	
	privateAttribute
	publicAttribute
	protectedAttribute
	packageAttribute
<hr/>	
	privateOperation()
	publicOperation()
	protectedOperation()
	packageOperation()

```
public class Visibility
{
    private int privateAttribute;
    public int publicAttribute;
    protected int protectedAttribute;
    int packageAttribute;

    private void privateOperation()
    public void publicOperation()
    protected void protectedOperation()
    void packageOperation()
}
```

Protected visibility



*protected properties
in the base class
are accessible to
a derived class
(subclass)*

Accessibility of inherited class properties

Name	Parent	Type
campaignCode	ECampaign	String
campaignTitle	ECampaign	String
dateClose	ECampaign	java.util.Date
dateDrawn	ECampaign	java.util.Date
numTickets	ECampaign	int
numTicketsSold	ECampaign	int
ticketBookSize	EBonusCampaign	int

Operation	Return type	Parent
computeTicketsSold	int	ECampaign
computeTicketsLeft	int	ECampaign
computeDaysLeft	int	ECampaign
getBookSize	int	EBonusCampaign

```
public class EBonusCampaign extends ECampaign
{
    private int ticketBookSize;

    public int getBookSize()
    {
        return 0;
    }

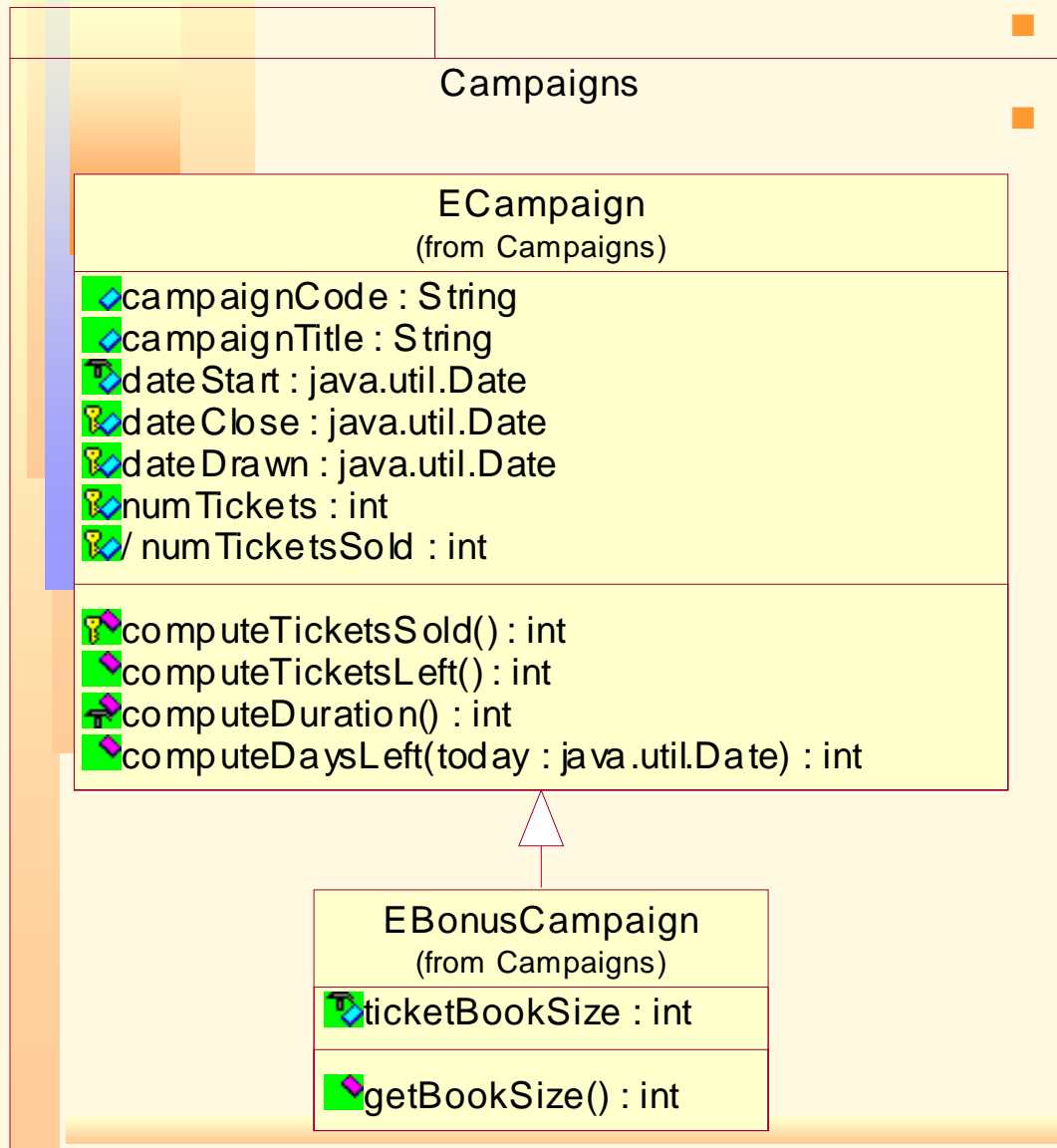
    public EBonusCampaign()
    {
        System.out.println("EBonusCampaign constructor");
    }

    public static void main(String[] args)
    {
        EBonusCampaign x = new EBonusCampaign();
        x.computeDuration();
        //The above will fail (x cannot access computeDuration)
    }
}
```

inheriting a property does not necessarily mean that the property is accessible with the objects of a derived class

private properties of the base class remain private to the base and are inaccessible to objects of the derived class

Package visibility



- *visible to all other classes in the package*
- *protected (and public) gives package access, but not vice versa*
 - *derived classes cannot access properties with package visibility if the derived and the base class are in different packages*

```
package Campaigns;
```

```
class EBonusCampaign extends
```

```
ECampaign
```

```
{
```

```
    int ticketBookSize;
```

```
    public int getBookSize()
```

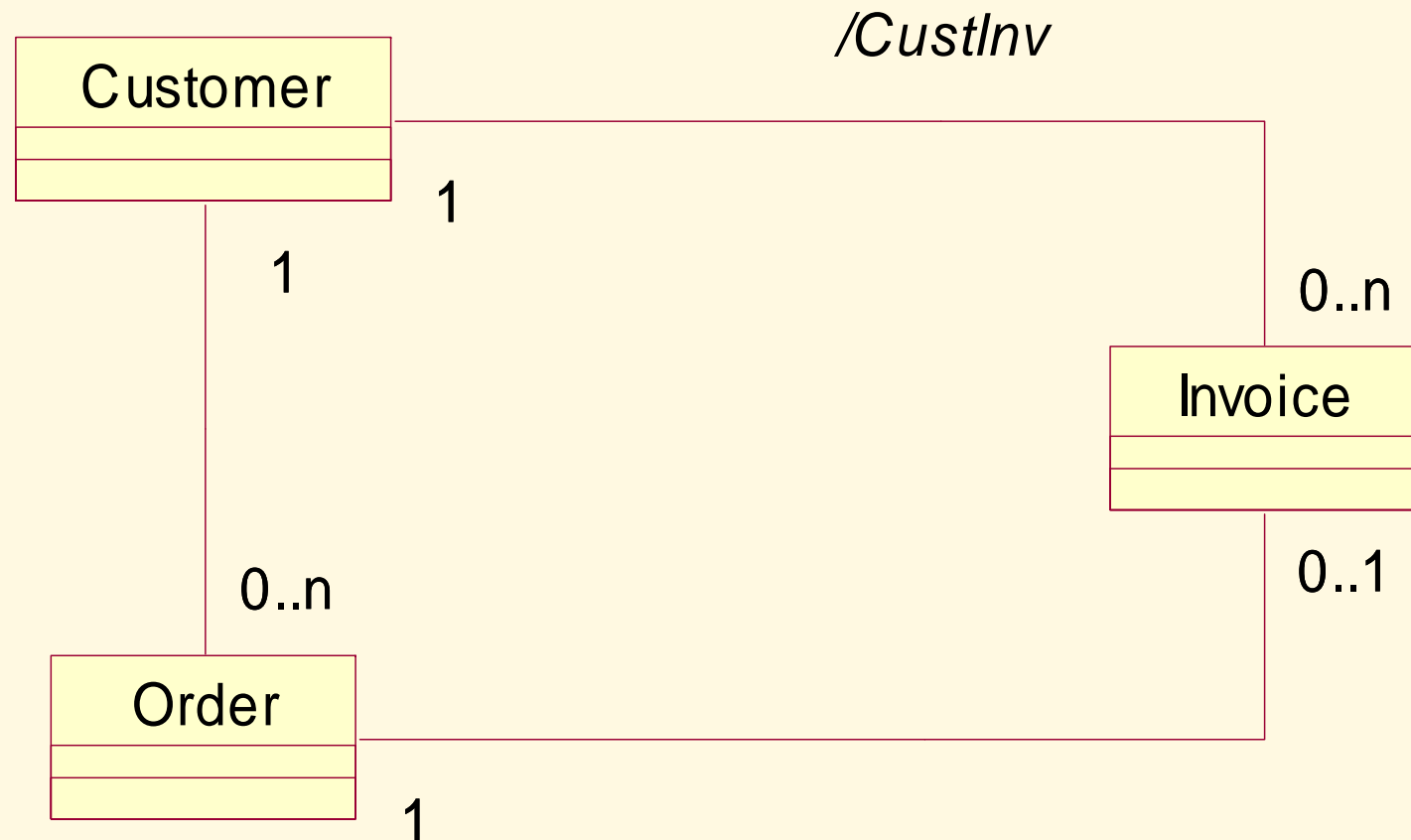
```
    {
```

```
        return 0;
```

```
    }
```

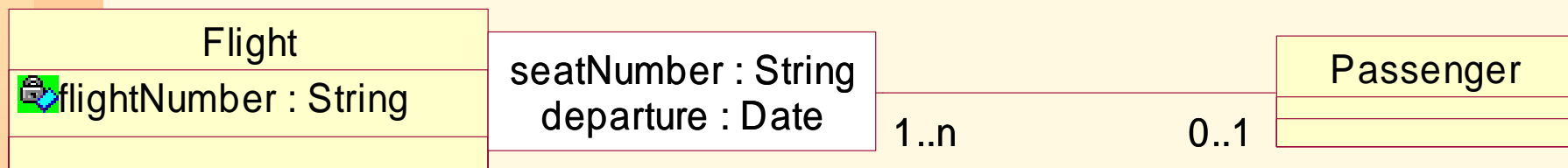
```
}
```

Derived association

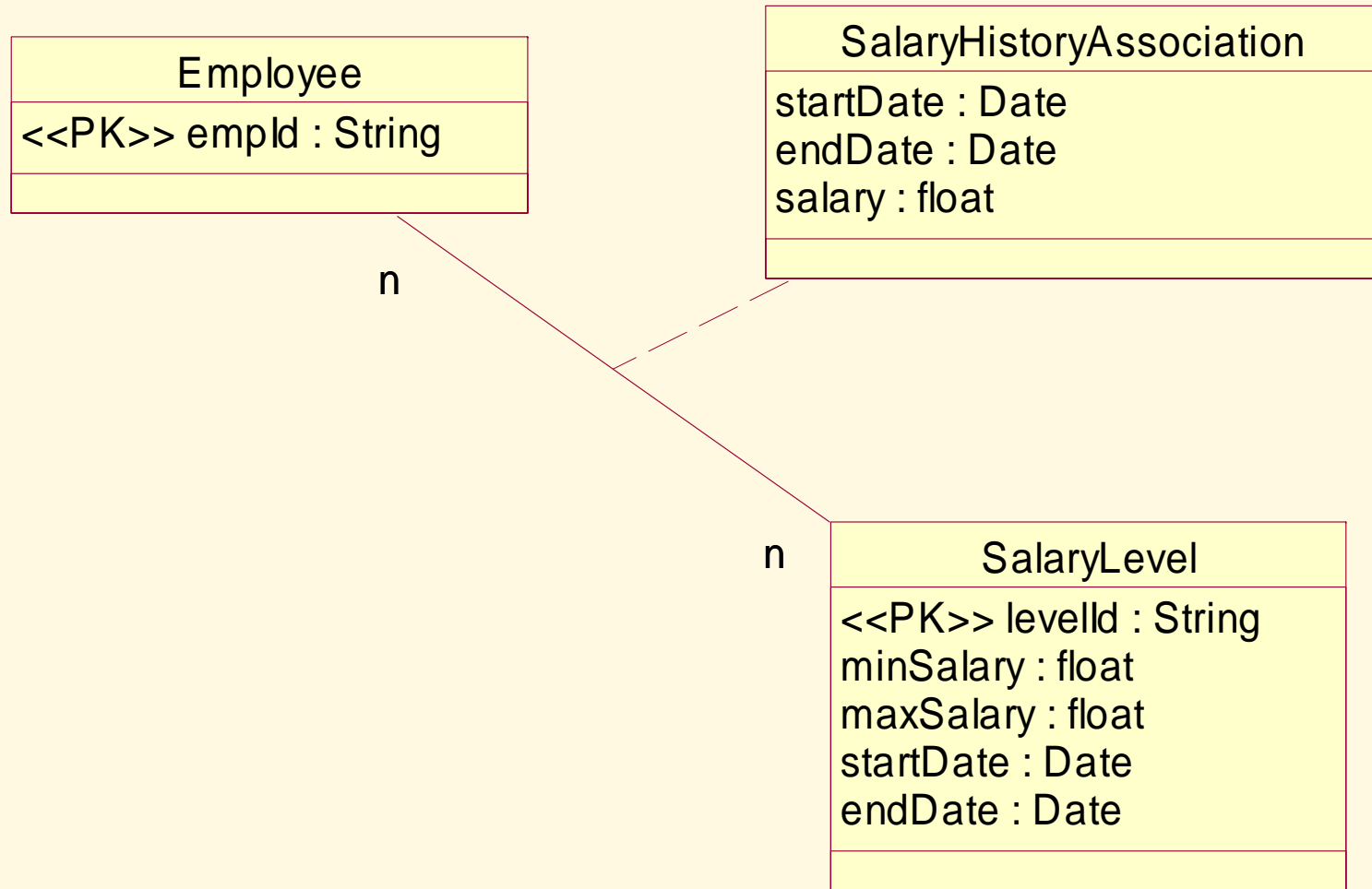


Qualified association

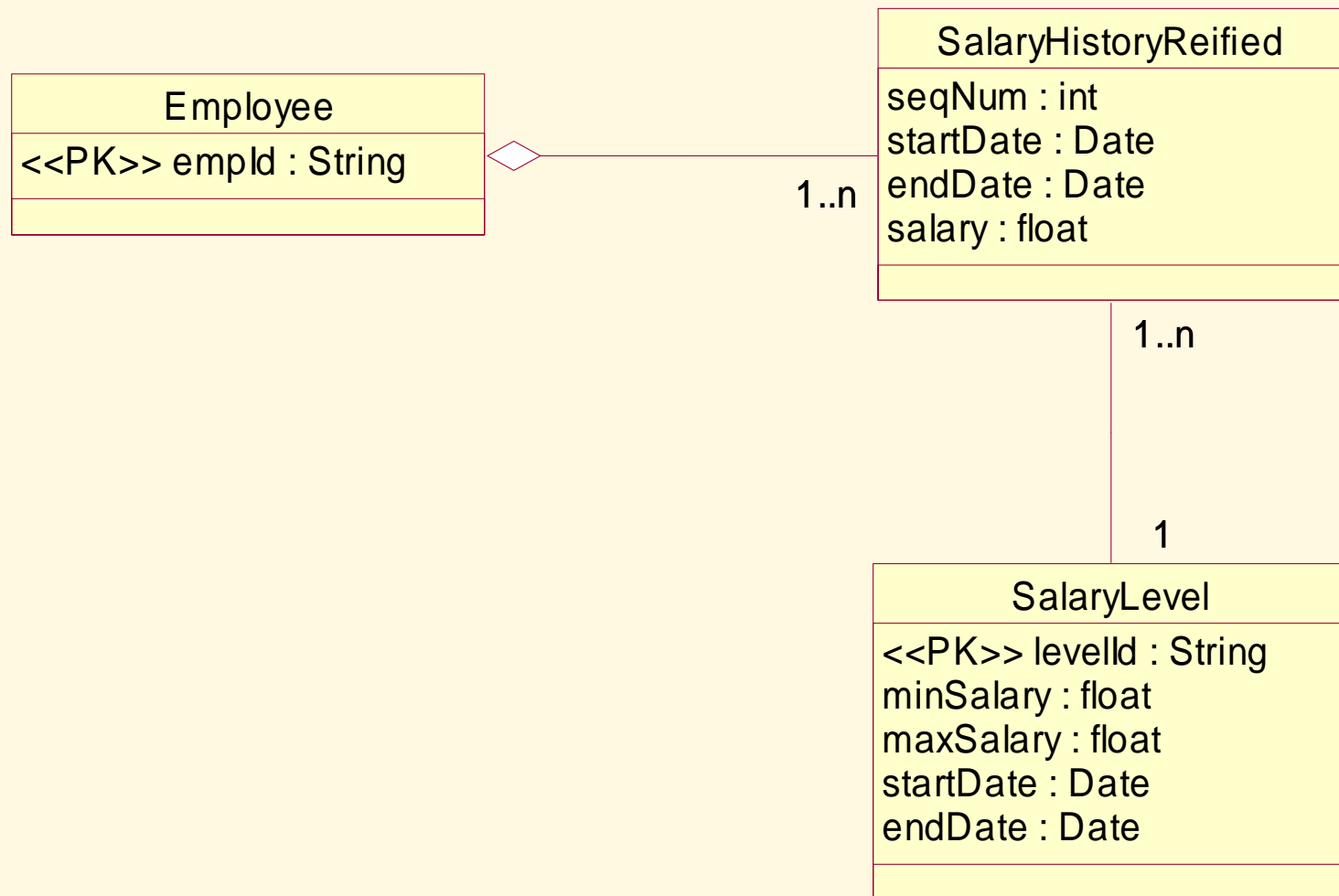
- A qualified association has an attribute compartment (a qualifier) on one end of a binary association (an association can be qualified on both ends but this is rare).
- The compartment contains one or more attributes that can serve as an index key for traversing the association from the qualified source class via the qualifier to the target class on the opposite association end



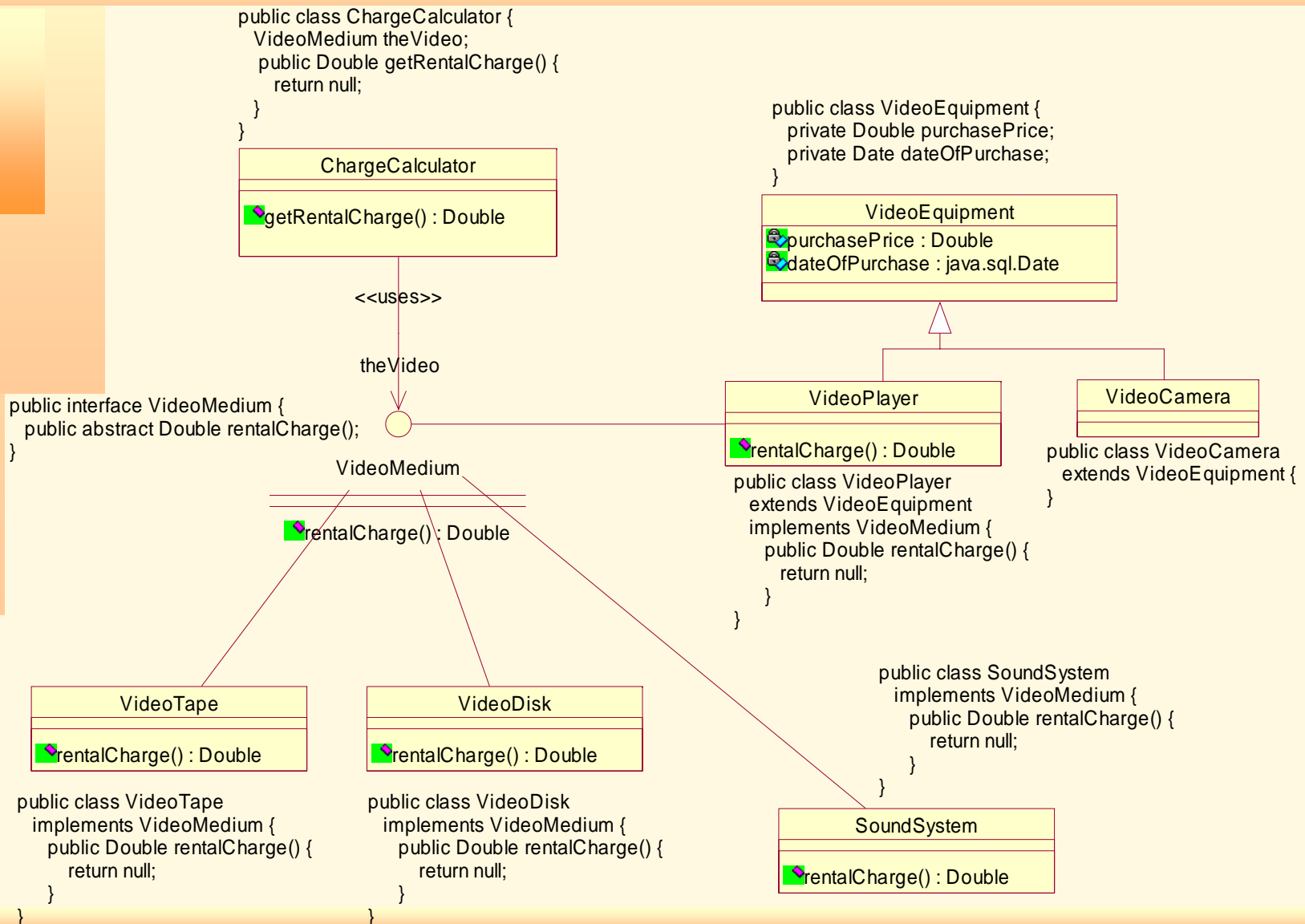
Association class



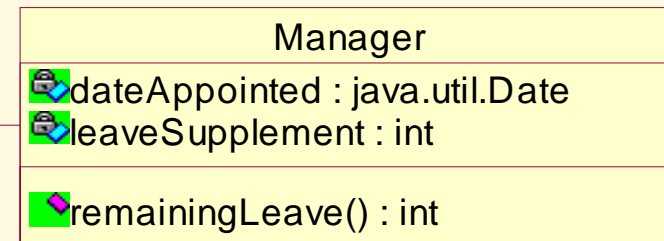
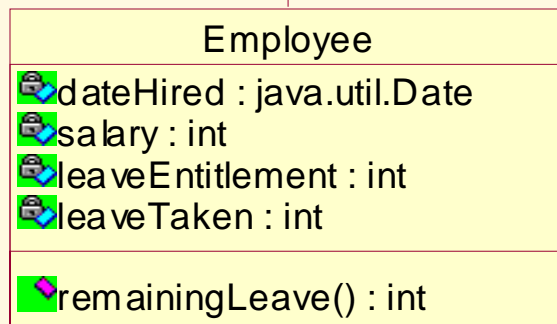
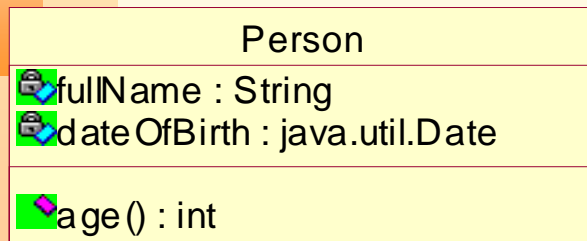
Replacing association class with a reified class



Interface and implementation inheritance revisited



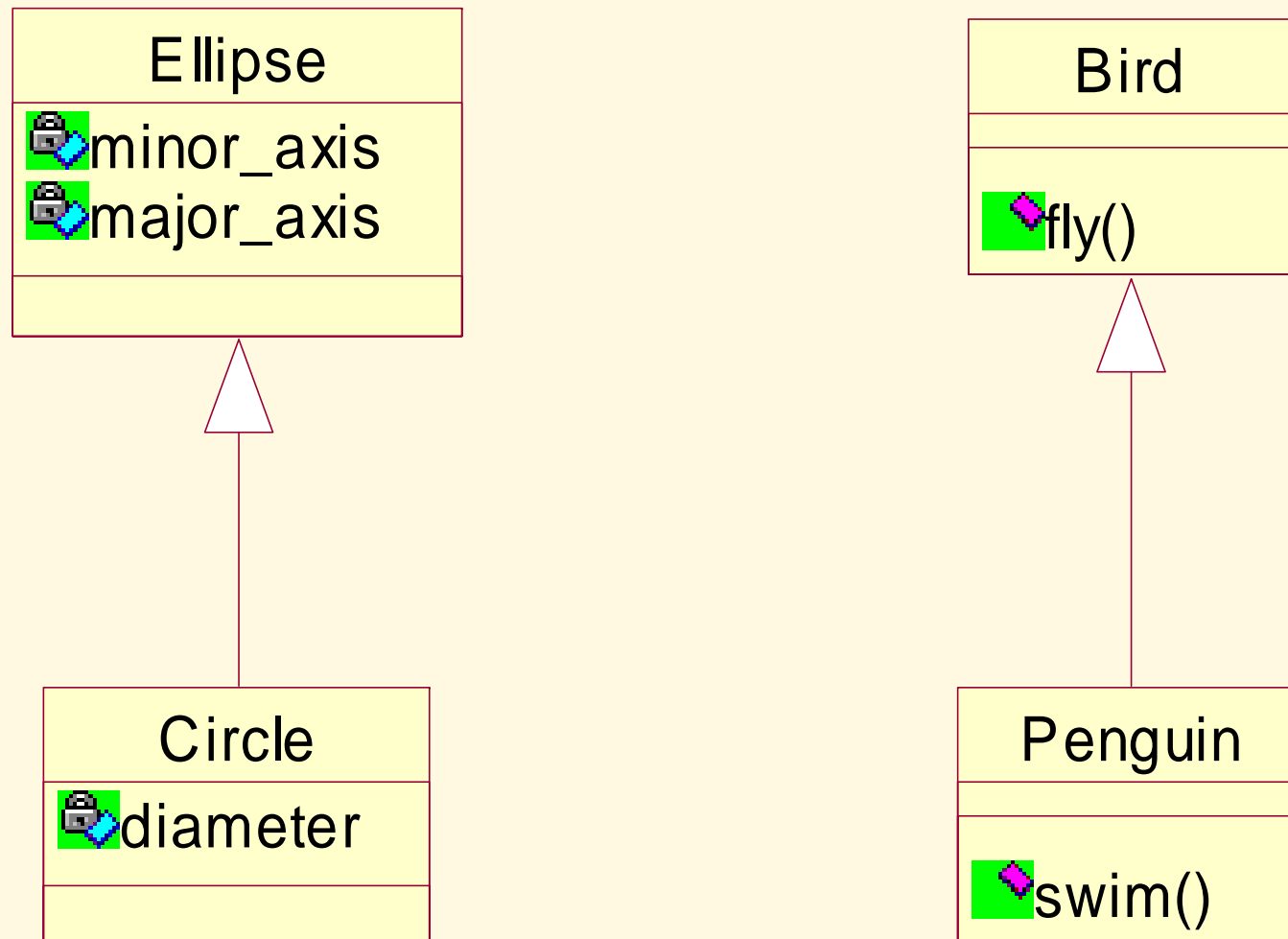
Extension inheritance



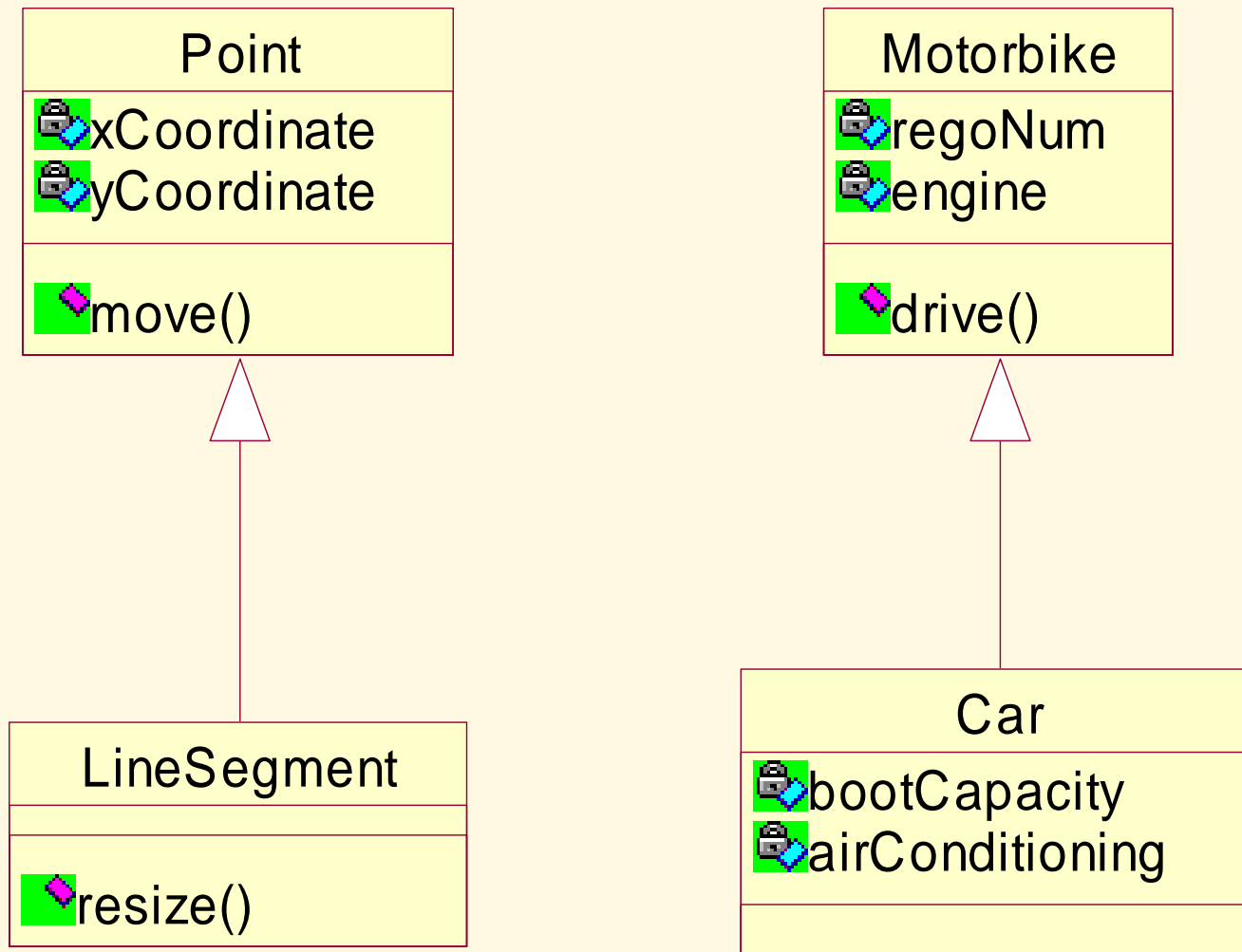
```
public class Manager extends Employee
{
    private Date dateAppointed;
    private int leaveSupplement;

    public int remainingLeave()
    {
        int mrl;
        mrl = super.remainingLeave() + leaveSupplement;
        return mrl;
    }
}
```

Restriction inheritance



Convenience inheritance



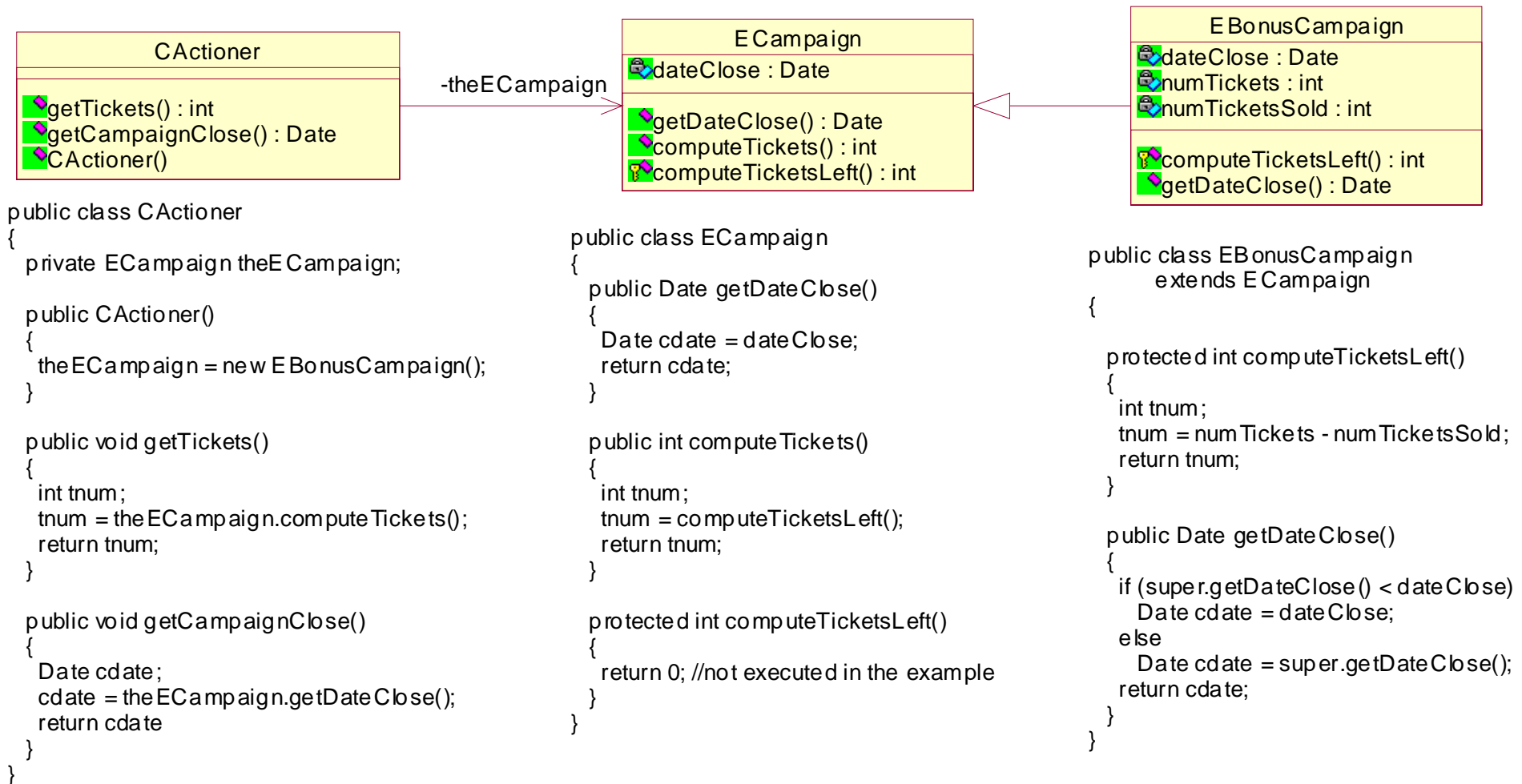
Fragile base class

- *Change in a superclass automatically affects the existing subclasses*
- *Changes on superclass' public interfaces are particularly troublesome*
 - *Changing the signature of a method.*
 - *Splitting the method into two or more new methods.*
 - *Joining existing methods into a larger method.*
- *'madness is inherited, you get it from your children'*

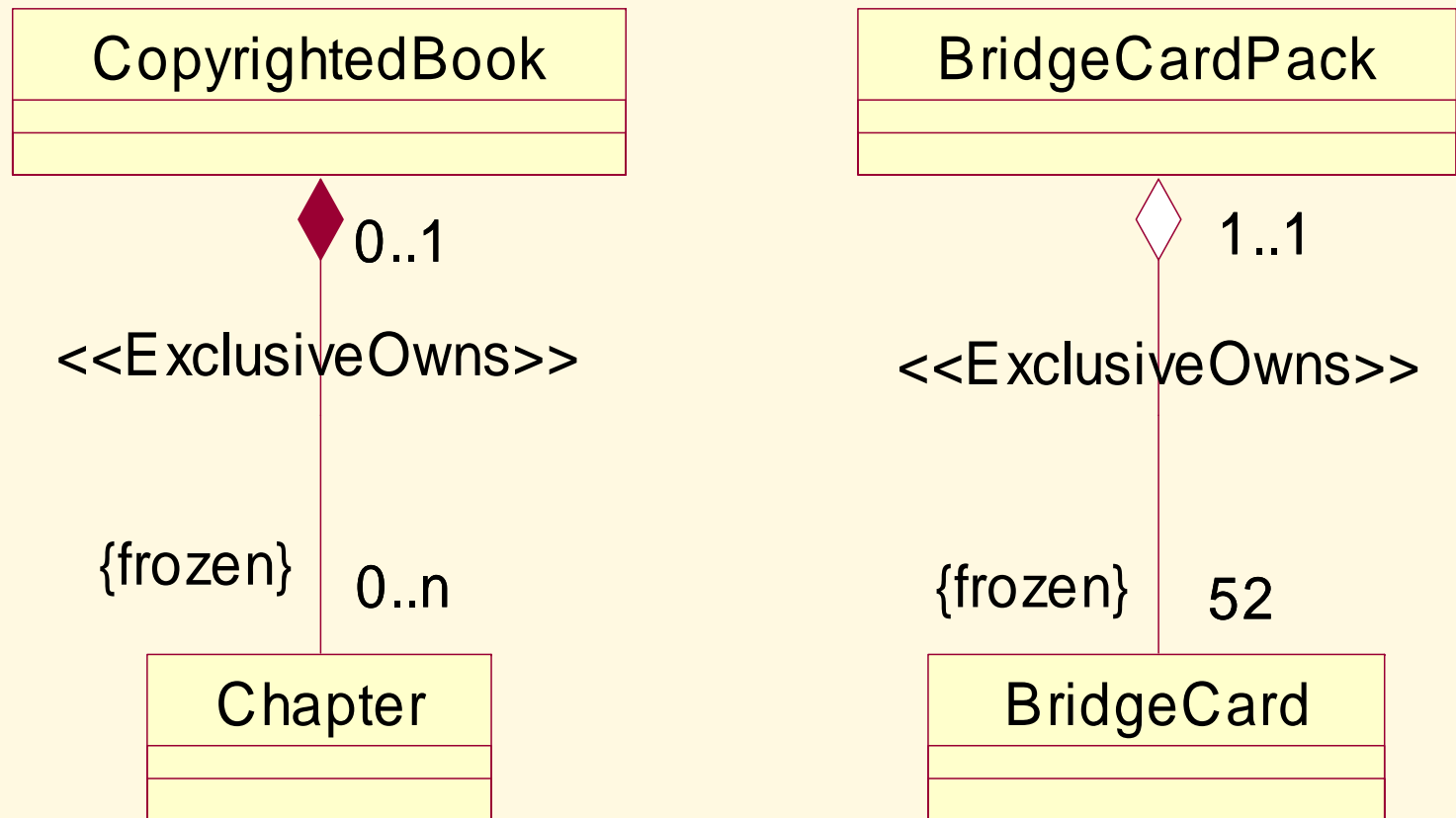
Reusing the superclass' code

- *The subclass can inherit the method implementation and introduce no changes to the implementation.*
- *The subclass can inherit the code and include it (call it) in its own method with the same signature.*
- *The subclass can inherit the code and then completely override it with a new implementation with the same signature.*
- *The subclass can inherit code that is empty (i.e. the method declaration is empty) and then provide the implementation for the method.*
- *The subclass can inherit the method interface only (i.e. the interface inheritance) and then provide the implementation for the method.*

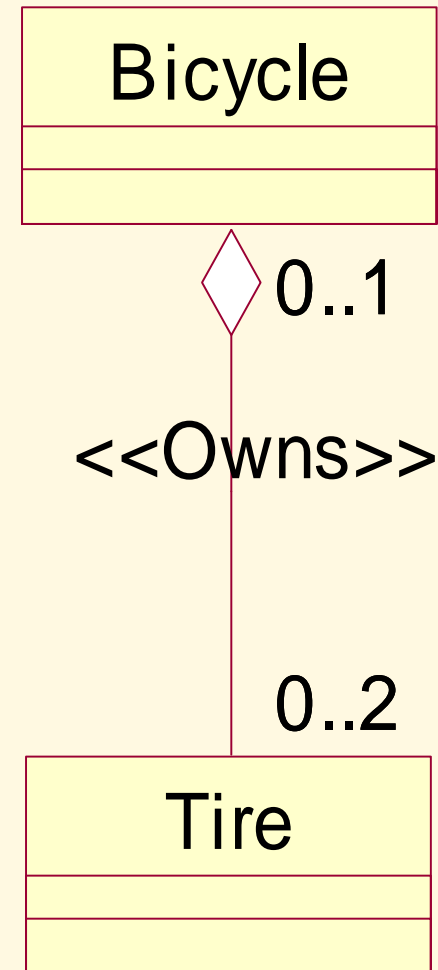
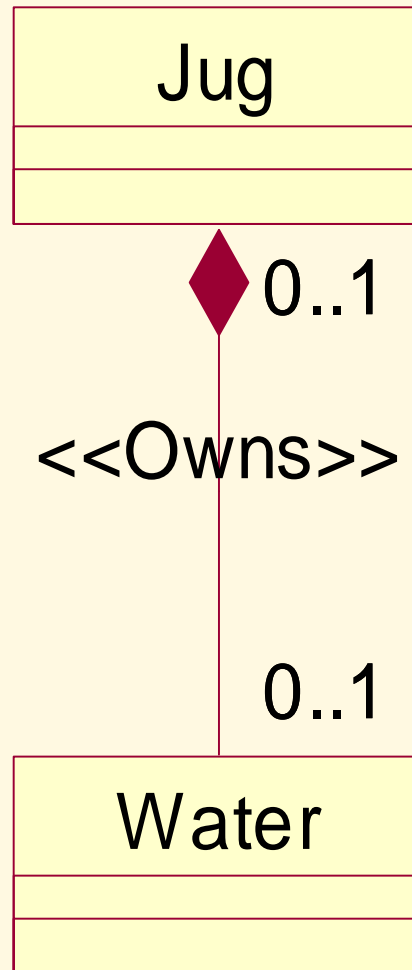
Overriding, down-calls and up-calls



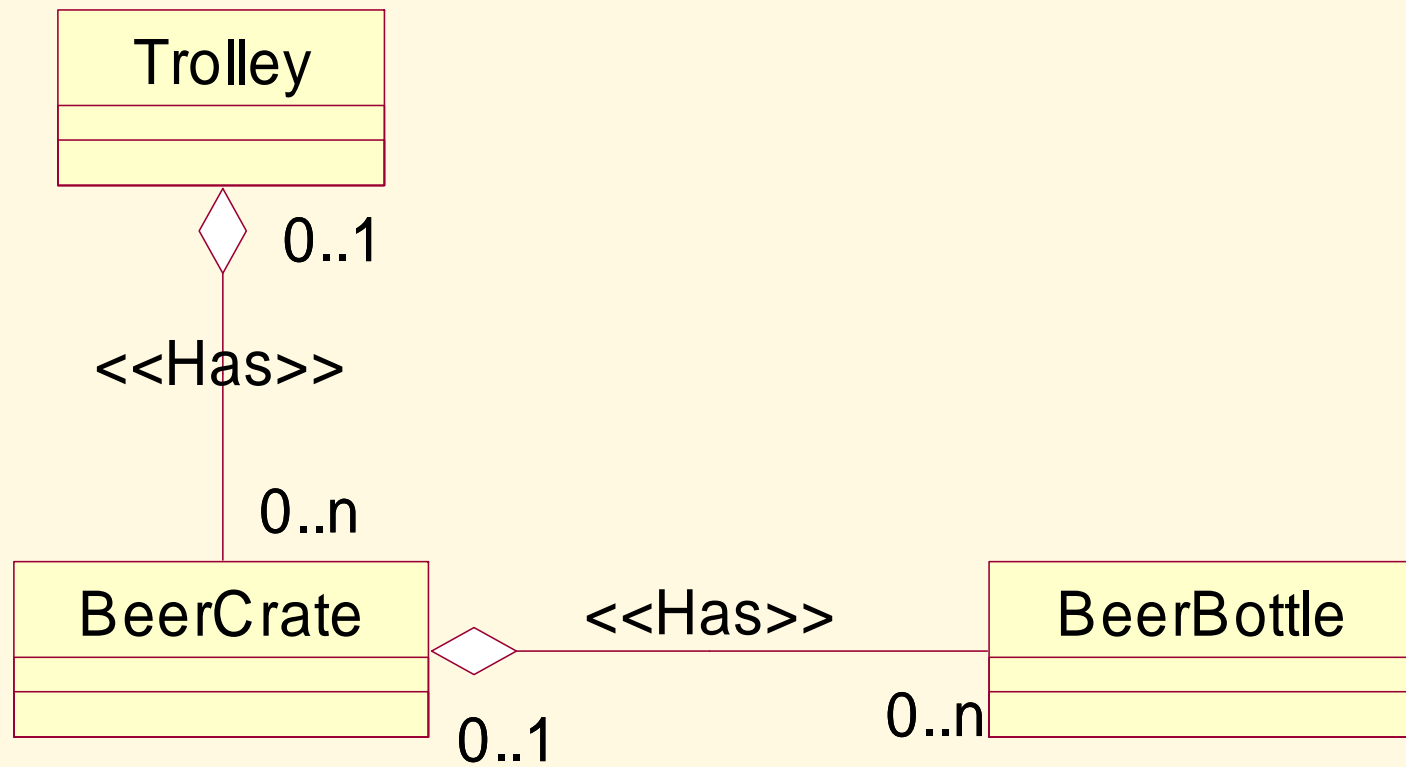
ExclusiveOwns aggregation



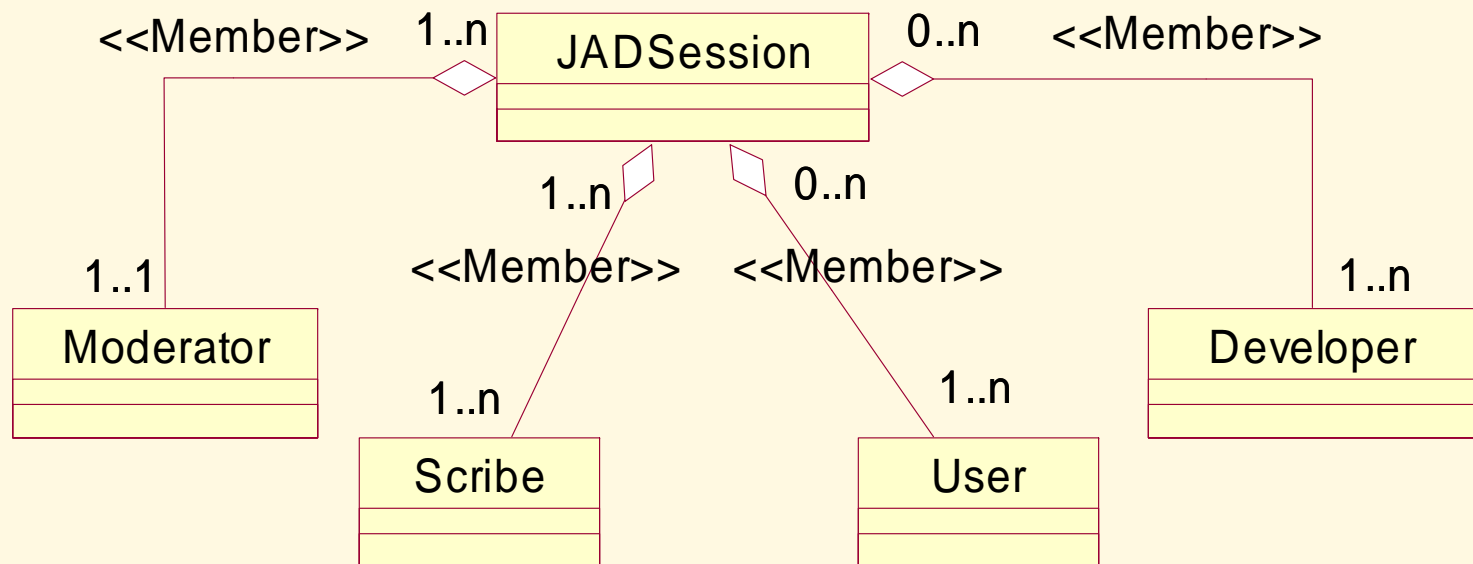
Owns aggregation



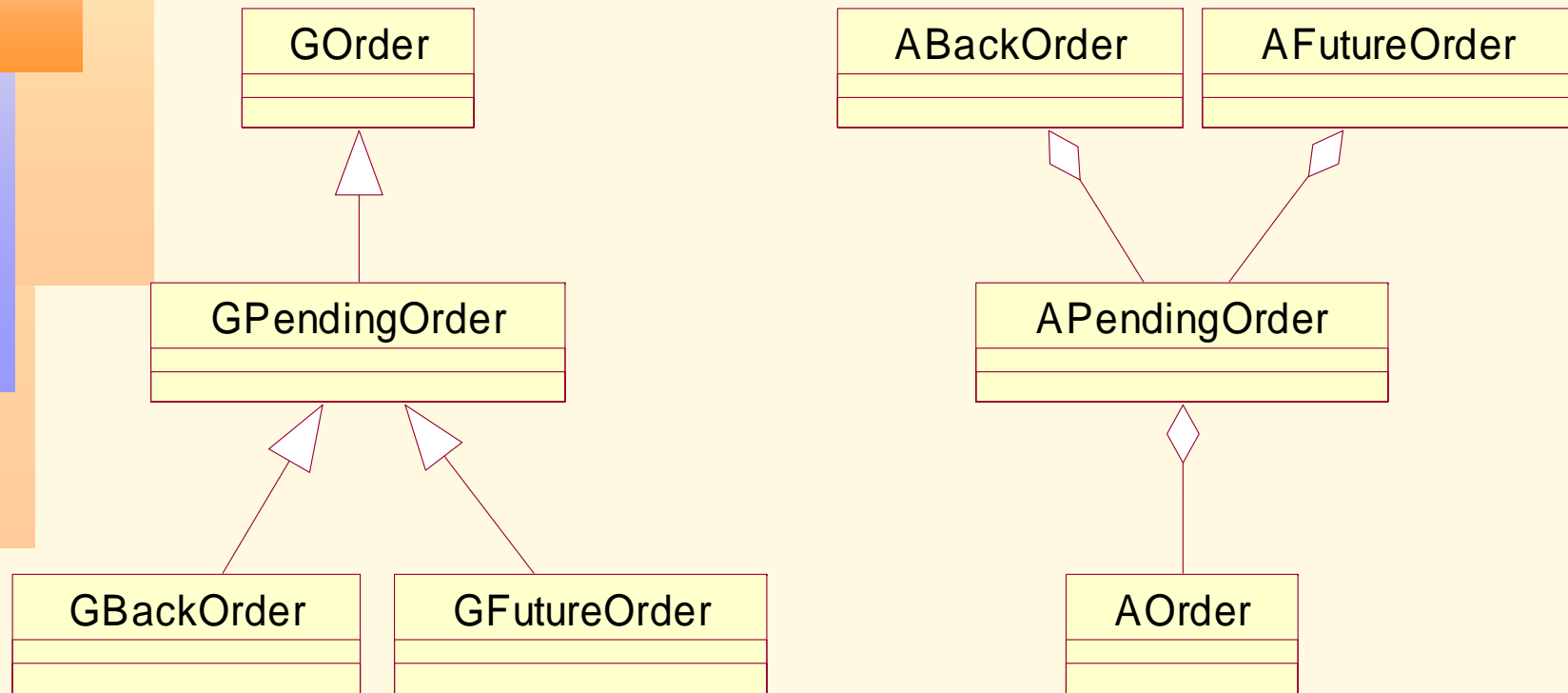
Has aggregation



Member aggregation



Generalization vs aggregation



Summary

- **Stereotypes** are the main extensibility technique of UML. In the extensibility task, they are assisted by **constraints** and **tags**.
- **Protected** visibility permits control of encapsulation within the inheritance structures.
- **Package** visibility comes handy where private visibility is too restrictive.
- One of the most intriguing aspects of class modeling is the choice between an **association class** and a **reified class**.
- The concept of **generalization** and **inheritance** is a double-edged sword in system modeling.
- The concept of aggregation and delegation is an important modeling alternative to generalization and inheritance.