

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

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 2

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

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 3

Stereotypes

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

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 4

Comments and constraints

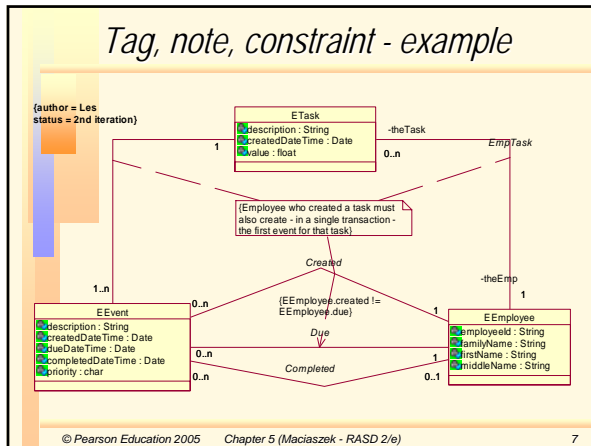
- **Comment** - text string attached to a model element
 - can be presented as a UML note
- **Constraint** - semantic relationship among model elements that specifies conditions and propositions that must be maintained as true
 - enclosed in braces {...}

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 5

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

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 6



Visibility and encapsulation

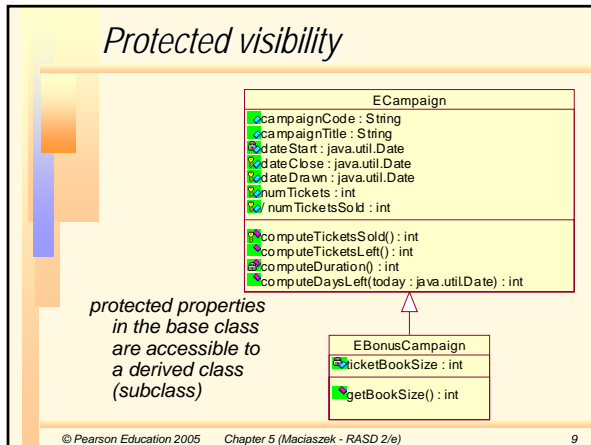
- + for public visibility
- for private visibility
- # for protected visibility
- ~ for package visibility

```

public class Visibility
{
    private Attribute
    public Attribute
    protected Attribute
    package Attribute

    private Operation()
    public Operation()
    protected Operation()
    package Operation()
}
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 8

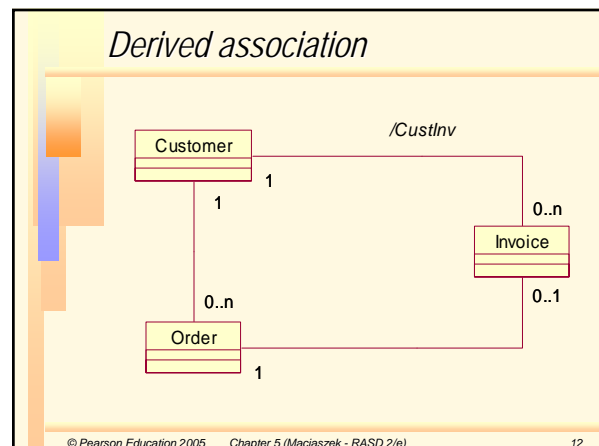
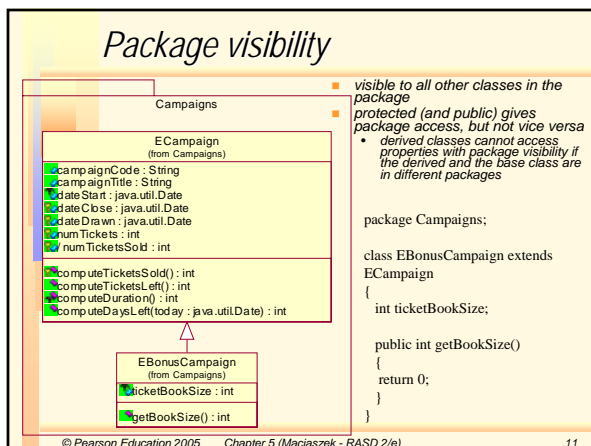


Accessibility of inherited class properties

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

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 10



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

```

classDiagram
    class Flight {
        flightNumber : String
        seatNumber : String
        departure : Date
    }
    class Passenger {
        departure : Date
    }
    Flight "1..n" -- "0..1" Passenger
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 13

Association class

```

classDiagram
    class Employee {
        emplid : String
    }
    class SalaryLevel {
        levelid : String
        minSalary : float
        maxSalary : float
        startDate : Date
        endDate : Date
    }
    class SalaryHistoryAssociation {
        startDate : Date
        endDate : Date
        salary : float
    }
    Employee "n" -- "n" SalaryLevel
    Employee -- SalaryHistoryAssociation
    SalaryLevel -- SalaryHistoryAssociation
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 14

Replacing association class with a reified class

```

classDiagram
    class Employee {
        emplid : String
    }
    class SalaryHistoryReified {
        seqNum : int
        startDate : Date
        endDate : Date
        salary : float
    }
    class SalaryLevel {
        levelid : String
        minSalary : float
        maxSalary : float
        startDate : Date
        endDate : Date
    }
    Employee "1" o-- "1..n" SalaryHistoryReified
    SalaryHistoryReified "1..n" -- "1" SalaryLevel
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 15

Interface and implementation inheritance revisited

```

classDiagram
    class ChargeCalculator {
        getRentalCharge() : Double
    }
    class VideoMedium {
        rentalCharge() : Double
    }
    class VideoTape {
        rentalCharge() : Double
    }
    class VideoDisk {
        rentalCharge() : Double
    }
    class VideoEquipment {
        purchasePrice : Double
        oIPurchase : java.sql.Date
    }
    class VideoPlayer {
        rentalCharge() : Double
    }
    class VideoCamera {
        rentalCharge() : Double
    }
    class SoundSystem {
        rentalCharge() : Double
    }
    ChargeCalculator <|-- VideoMedium
    VideoMedium <|-- VideoTape
    VideoMedium <|-- VideoDisk
    VideoEquipment <|-- VideoPlayer
    VideoEquipment <|-- VideoCamera
    VideoPlayer <|-- VideoTape
    VideoPlayer <|-- VideoDisk
    VideoDisk <|-- SoundSystem
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 16

Extension inheritance

```

classDiagram
    class Person {
        fullName : String
        birth : java.util.Date
        age : int
    }
    class Employee {
        dateHired : java.util.Date
        salary : int
        leaveEntitlement : int
        leaveTaken : int
        remainingLeave() : int
    }
    class Manager {
        dateAppointed : java.util.Date
        leaveSupplement : int
        remainingLeave() : int
    }
    Person <|-- Employee
    Employee <|-- Manager
    
```

```

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

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

```

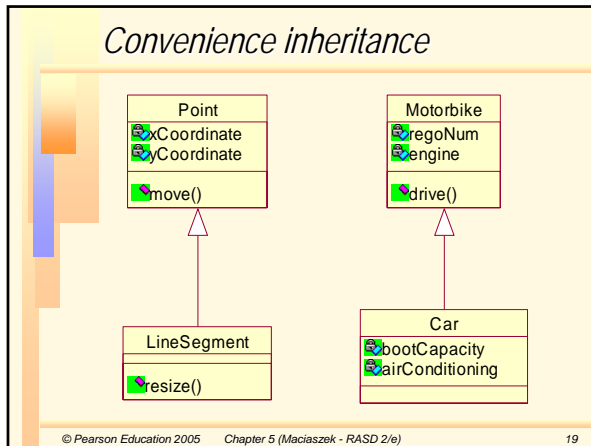
© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 17

Restriction inheritance

```

classDiagram
    class Ellipse {
        minor_axis
        major_axis
    }
    class Circle {
        diameter
    }
    class Bird {
        fly()
    }
    class Penguin {
        swim()
    }
    Ellipse <|-- Circle
    Bird <|-- Penguin
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 18



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'

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 20

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.

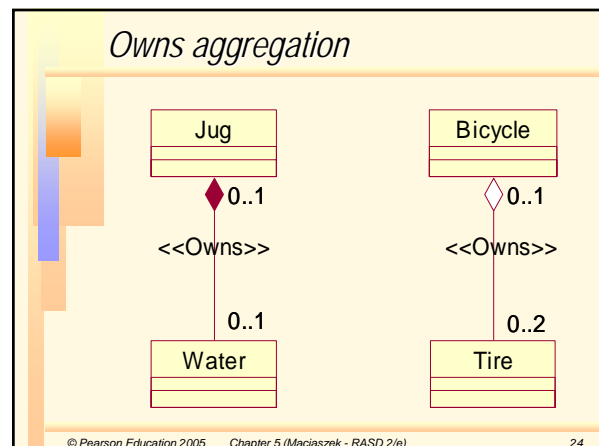
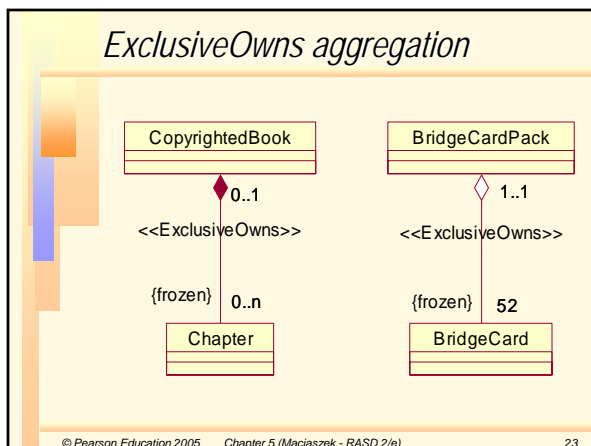
© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 21

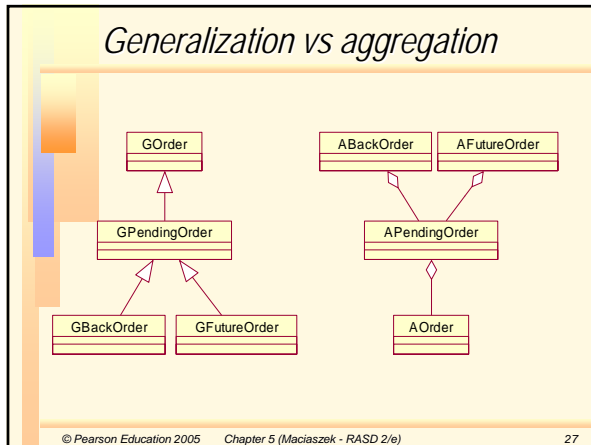
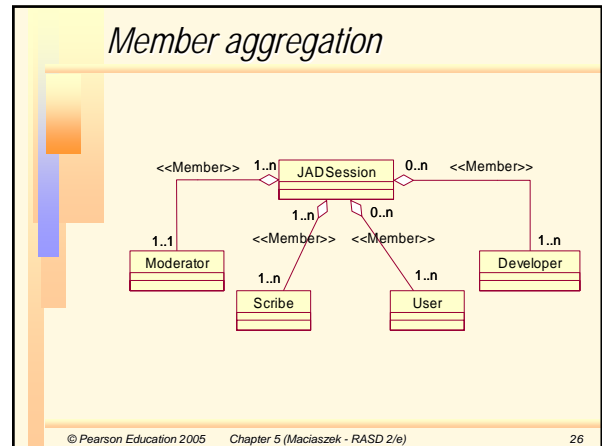
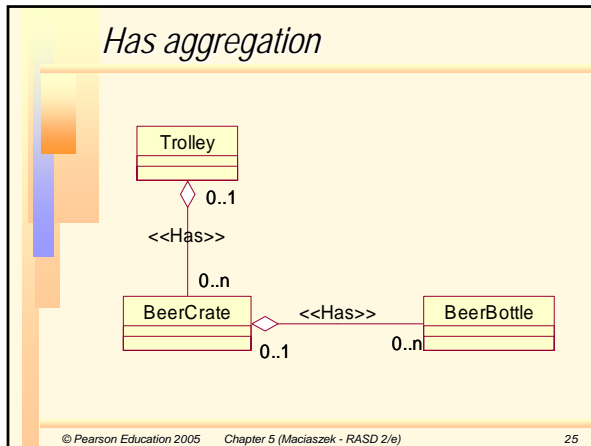
Overriding, down-calls and up-calls

```

classDiagram
    class CActioner {
        getTickets() : int
        getCampaignClose() : Date
        CActioner()
    }
    class ECampaign {
        dateClose : Date
        computeTickets() : int
    }
    class EBonusCampaign {
        numTicketsSold : int
        computeTicketsLeft() : int
        getDateClose() : Date
    }
    CActioner <|-- ECampaign
    ECampaign <|-- EBonusCampaign
    
```

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 22





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.

© Pearson Education 2005 Chapter 5 (Maciaszek - RASD 2/e) 28