

MACIASZEK, L.A. (2001): *Requirements Analysis and System Design. Developing Information Systems with UML*, Addison Wesley

Chapter 9
Program and Transaction Design

Copyright © 2000 by Addison Wesley
Version 1.0

Topics

- *Designing the Program*
- *Program Navigation*
- *Designing the Transaction*
- *Round-Trip Engineering*

(c) Addison Wesley Chapter 9 2

Designing the program

- *Design*
 - *Architectural design*
 - *Detailed design*
 - *Program design*
 - *One application program at a time*
 - *Extends the GUI design*
 - *Uses a database subschema*
 - *Defines database procedural parts - stored procedures*
 - *Spans the client and the server processes*

(c) Addison Wesley Chapter 9 3

Class cohesion and coupling

- **Class cohesion**
 - Degree of inner self-determination of the class
 - Measure of the strength of the class independence
 - One action, a single goal
 - The stronger the better
- **Class coupling**
 - Degree of connections between classes
 - Measures the class interdependence
 - The weaker the coupling – the better
- **Better cohesion induces worse coupling and vice versa**

(c) Addison Wesley Chapter 9 4

Cohesion and coupling heuristics

- **Two classes to either be not dependent on one another or one class to be only dependent on the public interface of another class**
- **Attributes and the related methods to be kept in one class**
- **A class to capture one and only one abstraction - unrelated information to be kept in separate classes**
- **The system intelligence to be distributed as uniformly as possible**

(c) Addison Wesley Chapter 9 5

Kinds of class coupling

- **X inherits from Y**
- **X has an attribute of class Y**
- **X has a template attribute with a parameter of class Y**
- **X has a method with an input argument of class Y**
- **X has a method with an output argument of class Y**
- **X knows of a global variable of class Y**
- **X knows of a method containing a local variable of class Y**
- **X is a friend of Y**

(c) Addison Wesley Chapter 9 6

Coupling and BCDE approach

- BCDE specifies four layers of classes
- Objects communicate within a layer and between the adjacent layers
- **Intra-layer coupling**
 - desirable
 - localizes software maintenance and evolution to individual layers
- **Inter-layer coupling**
 - to be minimized
 - communication interfaces to be carefully defined
- The Law of Demeter to be obeyed

(c) Addison Wesley Chapter 9 7

The Law of Demeter

- **Message target** can only be one of the following objects
 - 1 The method's object itself (i.e. this in C++ and Java, self and super in Smalltalk)
 - 2 An object that is an argument in the method's signature
 - 3 An object referred to by the object's attribute (including an object referred to within a collection of attributes)
 - 4 An object created by the method
 - 5 An object referred to by a global variable
- **The Strong Law of Demeter**
 - Rule 3 limited to attributes defined in the class - inherited attributes not to be used to identify the target object

(c) Addison Wesley Chapter 9 8

Accessor methods and mindless classes

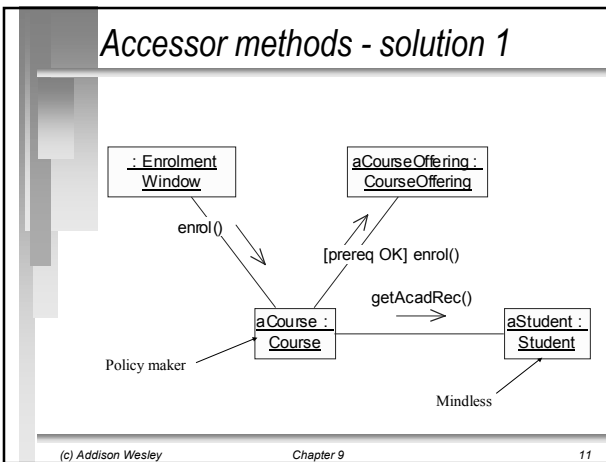
- **Accessor methods**
 - observers (*get*)
 - mutators (*set*)
- Object state can only be accessed through accessor methods in its interface
- **Mindless class**
 - puts too many accessor methods in its interface
 - allows other objects to freely view and modify its information content
 - sometimes difficult to avoid
 - "On an object-oriented farm there is an object-oriented milk. Should the object-oriented cow send the object-oriented milk the uncow_yourself message, or should the object-oriented milk send the object-oriented cow the unmilkm_yourself message?"

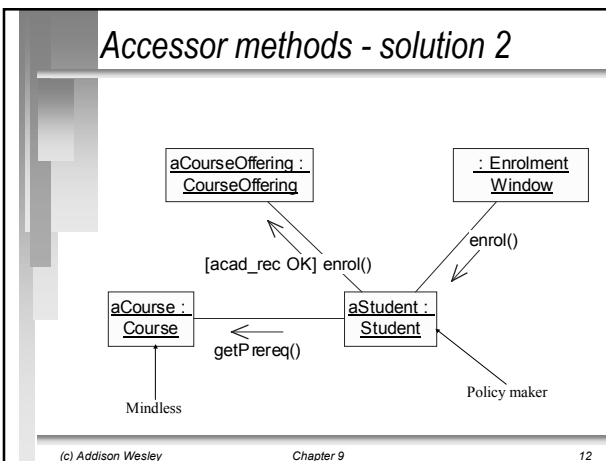
(c) Addison Wesley Chapter 9 9

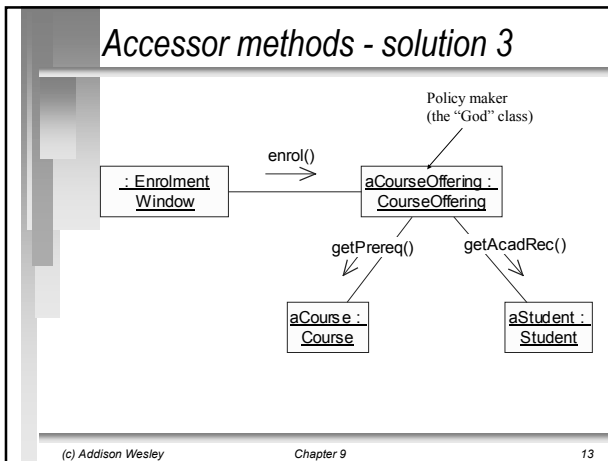
Accessor methods - example

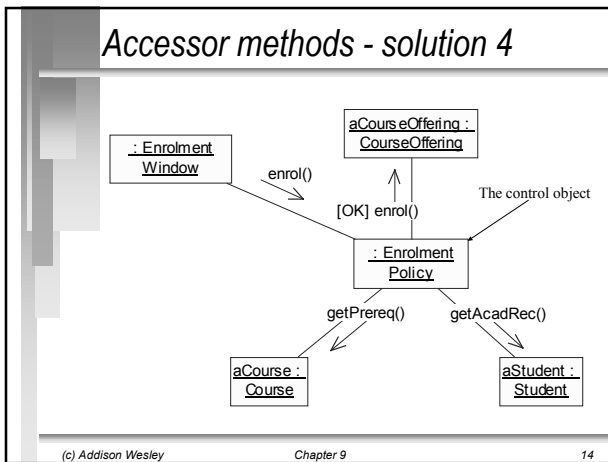
- Add a student to a course offering.
- Check
 - what are the prerequisite courses for the course offering
 - if the student satisfies the prerequisites
- Consider that
 - the `enrol()` message is sent by the boundary object `:EnrolmentWindow`
 - Three entity classes are involved – `CourseOffering`, `Course`, and `Student`
- At least four solutions possible

(c) Addison Wesley Chapter 9 10









Mixed-instance cohesion

- The consequence of the lack of **dynamic classification**
- A class with **mixed-instance cohesion**
 - “has some features that are undefined for some objects of the class”
 - features (methods and attributes) do not apply to all object of the class
 - e.g. not all objects of the *Employee* class get allowance; only *Manager* objects do (i.e. employees who are managers).

(c) Addison Wesley Chapter 9 15

Mixed-instance cohesion

```

classDiagram
    class Student {
        +current_sem_credit_points : Integer
    }
    class PartTimeStudent {
        +evening_preference : Boolean
        +payExtraFee(crs_off)
    }
    class FullTimeStudent
    Student <|-- PartTimeStudent
    Student <|-- FullTimeStudent
  
```

- This design has **no mixed-instance cohesion** provided that
 - extra fees are paid even if a part time student elects to take daytime course offerings

(c) Addison Wesley Chapter 9 16

Mixed-instance cohesion

- This design eliminates mixed-instance cohesion even if
 - extra fees are not paid by a part time student who elects to take daytime course offerings

```

classDiagram
    class Student {
        +current_sem_credit_points : Integer
    }
    class PartTimeStudent {
    }
    class FullTimeStudent
    class EveningPrefPartTimeStudent {
        +payExtraFee(crs_off)
    }
    class DayPrefPartTimeStudent
    Student <|-- PartTimeStudent
    Student <|-- FullTimeStudent
    PartTimeStudent <|-- EveningPrefPartTimeStudent
    PartTimeStudent <|-- DayPrefPartTimeStudent
  
```

(c) Addison Wesley Chapter 9 17

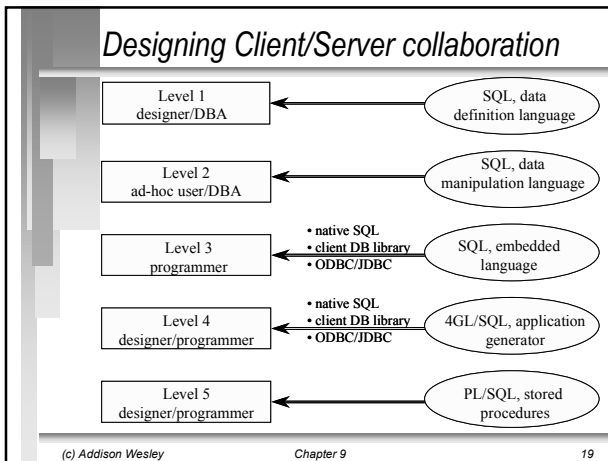
Mixed-instance cohesion

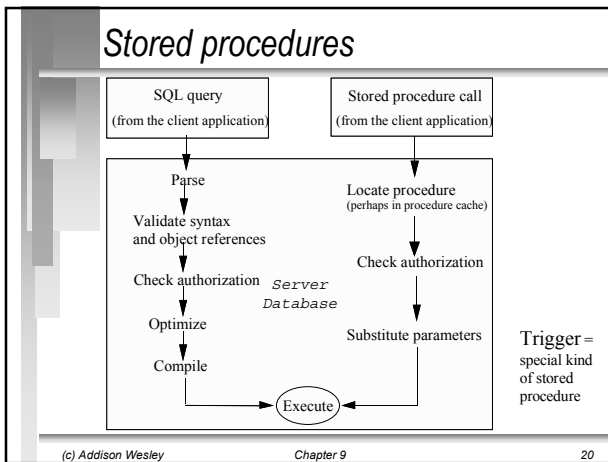
- But what if a DayPrefPartTimeStudent is forced to take an evening course offering because there are no more places available in daytime course offerings?
- Perhaps, some other fee would then apply. Should we specialize further to derive a class UnluckyDayPrefPartTimeStudent and avoid the mixed-instance cohesion again?
- We may have to use the IF statement in the code:

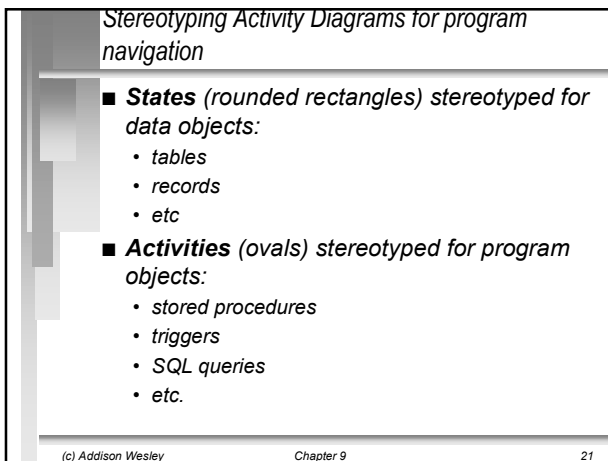
```

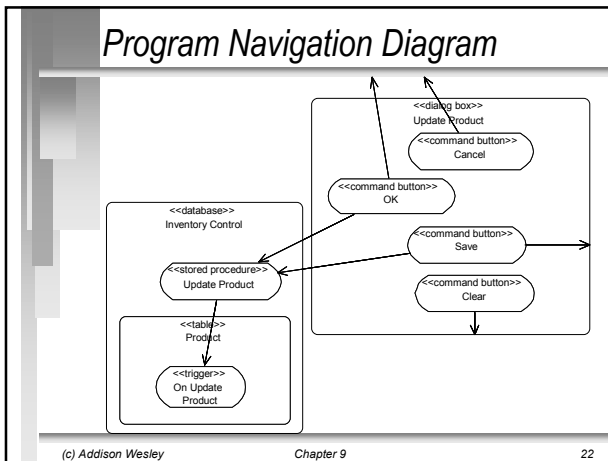
method payExtraFee(crs_off) for the class PartTimeStudent
  if evening_preference = "False"
    return
  else
    do it
  end method
  
```

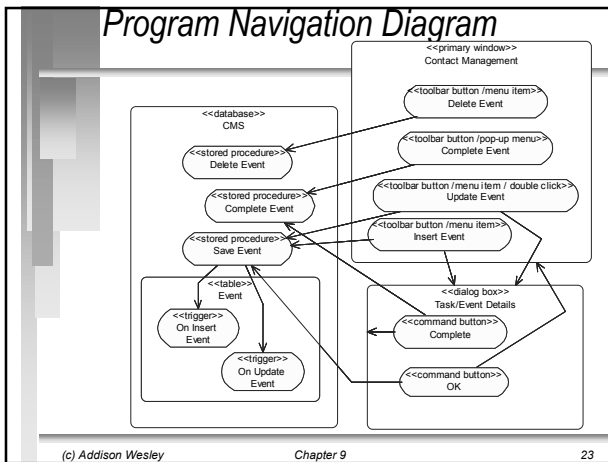
(c) Addison Wesley Chapter 9 18











Designing the transaction

- **Transaction**
 - logical unit of work
 - unit of database consistency
 - atomic - all results committed to DB or rolledback to original DB state
- **Transaction Manager**
 - database recovery
 - concurrency control
- **Short transaction - conventional DB business applications**

(c) Addison Wesley Chapter 9 24

Pessimistic concurrency control

- Locks on data objects:
 - Exclusive (write) lock** – other transactions must wait
 - Update (write intent) lock** – other transactions can read
 - Read (shared) lock** – other transactions can read and possibly obtain an update lock on the object
 - No lock** – other transactions can update an object

(c) Addison Wesley Chapter 9 25

Levels of transaction isolation

- Between concurrently executing transactions:
 - Dirty read possible** – transaction t1 modified an object but it has not committed yet; transaction t2 reads the object
 - Nonrepeatable read possible** – t1 has read an object; t2 updates the object; t1 reads the same object again
 - Phantom possible** – t1 has read a set of objects; t2 inserts a new object to the set; t1 repeats the read operation
 - Repeatable read** – t1 and t2 can still execute concurrently but the interleaved execution of these two transactions will produce the same results as if the transactions executed one at a time (**serializable execution**)

(c) Addison Wesley Chapter 9 26

Automatic recovery

- Rollback
- Rollforward
- Checkpoint - forcing changes to DB
- Backup recovery

Recovery after failure:
 t1 - rollforward (redo)
 t2 - rollback
 t3 - rollforward
 t4 - rollback
 t5 - no action

(c) Addison Wesley Chapter 9 27

Programmable recovery

- **Undo from user mistake**
 - *compensating transaction may be necessary*
- **Savepoint**
 - *divides a transaction into smaller parts*
 - *allows rollback to a named savepoint*
- **Trigger rollback**
 - *special kind of savepoint*
 - *only the trigger is rolled back and the transaction can take a remedial action*

(c) Addison Wesley Chapter 9 28

Long transaction

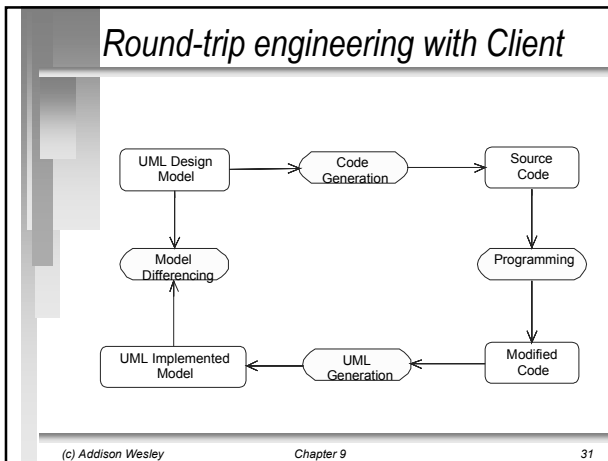
- *In workgroup computing applications*
- *Facilitate user cooperation whereas the purpose of short transactions is to isolate the users*
- *Can span computer sessions*
- **Require**
 - *version management*
 - *collaborative concurrency control*
 - *no automatic rollbacks*
- *Short transactions still required during check-in/check-out operations*

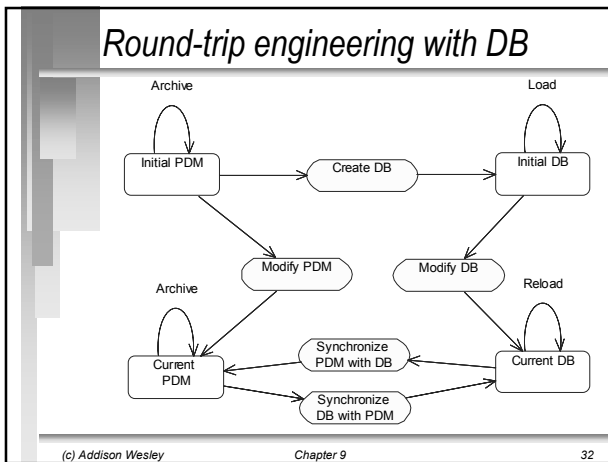
(c) Addison Wesley Chapter 9 29

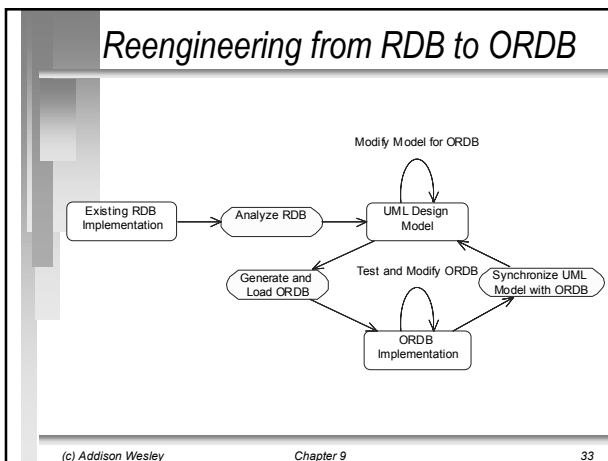
Round-trip engineering

- **The combination of**
 - *forward engineering*
 - *generation of code from the design*
 - *reverse engineering*
 - *recovery of design from the code*
- **Required for**
 - *client application programs*
 - *server database programs*

(c) Addison Wesley Chapter 9 30







Summary

- *The coupling and cohesion principles can be achieved through the **Law of Demeter***
- *Excessive use of **accessor methods** can lead to **mindless classes***
- ***Mixed-instance cohesion** sometimes necessary*
- *Five levels of **SQL interfaces***
- ***Program Navigation Diagrams** extend Window Navigation Diagrams*
- *Conventional DB applications require **short transactions***
- *Some new DB applications work in **long transactions***
- *Round-trip engineering applied to **client programs** and to **database programs***

(c) Addison Wesley Chapter 9 34
