Object-Oriented Software Engineering

WCB/McGraw-Hill, 2008

THE DESIGN WORKFLOW

Overview

- Object-oriented design
- Object-oriented design: The elevator problem case study
- Object-oriented design: The MSG Foundation case study
- The design workflow
- The test workflow: Design
- Formal techniques for detailed design
- Real-time design techniques

Overview (contd)

- CASE tools for design
- Metrics for design
- Challenges of the design workflow

Design Techniques

- Operation-oriented design
  - The emphasis is on the operations
  - Weakness: The data are of secondary importance
- Data-oriented design
  - The emphasis is on the data
  - Weakness: The operations are of secondary importance
- Object-oriented design
  - Gives equal weight to data and operations

12.1 Object-Oriented Design

- Aim
  - Design the product in terms of the classes extracted during the analysis workflow
- If we are using a language without inheritance (e.g., C, Ada 83)
  - Use abstract data type design
- If we are using a language without a type statement (e.g., FORTRAN, COBOL)
  - Use data encapsulation

Object-Oriented Design Steps

- OOD consists of two steps:

- Step 1. Complete the class diagram
  - Determine the formats of the attributes
  - Assign each method, either to a class or to a client that sends a message to an object of that class

- Step 2. Perform the detailed design

Object-Oriented Design Steps (contd)

- Step 1. Complete the class diagram
  - The formats of the attributes can be directly deduced from the analysis artifacts
- Example: Dates
  - U.S. format `mm/dd/yyyy`
  - European format `dd/mm/yyyy`
  - In both instances, 10 characters are needed
- The formats could be added during analysis
  - To minimize rework, *never* add an item to a UML diagram until strictly necessary

Object-Oriented Design Steps (contd)

- Assign each method, either to a class or to a client that sends a message to an object of that class
- Principle A: Information hiding
- Principle B: If an operation is invoked by many clients of an object, assign the method to the object, not the clients
- Principle C: Responsibility-driven design

Object-Oriented Design Steps (contd)

- Step 2. Perform the detailed design
  - Design each class in detail
- Select specific algorithms
- Choose data structures

Detailed Design of Method `find` of Class `Mortgage`

Class name	**Mortgage**
Method name	find
Return type	**boolean**
Input argument(s)	String findMortgageID
Output argument(s)	None
Error messages	If file not found, prints message ***** Error: Mortgage.find () *****
Files accessed	mortgage.dat
Files changed	None
Methods invoked	None
Narrative	Method find locates a given mortgage record if it exists. It returns true if the mortgage is located, otherwise false.

Figure 12.1

Detailed Design in PDL

- We can also write a detailed design in a program description language (PDL)
  - Formerly called pseudocode
- Next two slides:
- Java-like PDL description of detailed design of
  - Method `computeEstimatedFunds` of class **`EstimatedFundsForWeek`**
  - Method `totalWeeklyNetPayments` of class **`Mortgage`**

Method `EstimatedFundsForWeek.computeEstimatedFunds`


```
public static void computeEstimatedFunds( )
This method computes the estimated funds available for the week.
{
   float expectedWeeklyInvestmentReturn;          (expected weekly investment return)
   float expectedTotalWeeklyNetPayments = (float) 0.0;
                                                  (expected total mortgage payments
                                                  less total weekly grants)
   float estimatedFunds = (float) 0.0;            (total estimated funds for week)
Create an instance of an investment record.
   Investment inv = new Investment ( );
Create an instance of a mortgage record.
   Mortgage mort = new Mortgage ( );
Invoke method totalWeeklyReturnOnInvestment.
   expectedWeeklyInvestmentReturn = inv.totalWeeklyReturnOnInvestment ( );
Invoke method expectedTotalWeeklyNetPayments          (see Figure 12.3)
   expectedTotalWeeklyNetPayments = mort.totalWeeklyNetPayments ( );
Now compute the estimated funds for the week.
   estimatedFunds = (expectedWeeklyInvestmentReturn
         – (MSGApplication.getAnnualOperatingExpenses ( ) / (float) 52.0)
         + expectedTotalWeeklyNetPayments);

Store this value in the appropriate location.
   MSGApplication.setEstimatedFundsForWeek (estimatedFunds);
}  // computeEstimatedFunds
```

Figure 12.2

Method `Mortgage.totalWeeklyNetPayments`


```
public float totalWeeklyNetPayments ( )
This method computes the net total weekly payments made by the mortgagees, that is, the expected total weekly
mortgage amount less the expected total weekly grants.
{
   File mortgageFile = new File ("mortgage.dat");          (file of mortgage records)
   float expectedTotalWeeklyMortgages = (float) 0.0;       (expected total weekly mortgage payments)
   float expectedTotalWeeklyGrants = (float) 0.0;          (expected total weekly grants)
   float capitalRepayment;                                 (capital repayment)
   float interestPayment;                                  (interest payment)
   float escrowPayment;                                    (escrow payment)
   float tempMortgage;                                     (temporary value)
   float maximumPermittedMortgagePayment;                  (maximum amount the couple may pay)
Open the file of mortgages, name it inFile, and read each element in turn.
   {
      read (inFile);
Compute the capital repayment, interest payment, and escrow payment for this mortgage.
      capitalRepayment = price / NUMBER_OF_MORTGAGE_PAYMENTS;
      interestPayment = mortgageBalance * INTEREST_RATE / WEEKS_IN_YEAR ;
      escrowPayment = (annualPropertyTax + annualInsurancePremium) / WEEKS_IN_YEAR;
First assume that the couple can pay the mortgage in full, without a grant.
      tempMortgage = capitalRepayment + interestPayment + escrowPayment;
Add this amount to the running total of mortgage payments.
      expectedTotalWeeklyMortgages += tempMortgage;
Now determine how much the couple can actually pay.
      maximumPermittedMortgagePayment = currentWeeklyIncome *
               MAXIMUM_PERC_OF_INCOME;
If a grant is needed, add the grant amount to the running total of grants.
      if (tempMortgage > maximumPermittedMortgagePayment)
         expectedTotalWeeklyGrants += tempMortgage − maximumPermittedMortgagePayment;
   }
Close the file of mortgages.  Return the total expected net payments for the week.
   return (expectedTotalWeeklyMortgages − expectedTotalWeeklyGrants);
}  // totalWeeklyNetPayments
```

Figure 12.3

12.2 Object-Oriented Design: Elevator Problem

- Step 1. Complete the class diagram
- Assign the operations to the class diagram

OOD: Elevator Problem Case Study (contd)

- CRC card

CLASS **Elevator Controller Class**
RESPONSIBILITY 1. Send message to **Elevator Button Class** to turn on button 2. Send message to **Elevator Button Class** to turn off button 3. Send message to **Floor Button Class** to turn on button 4. Send message to **Floor Button Class** to turn off button 5. Send message to **Elevator Class** to move up one floor 6. Send message to **Elevator Class** to move down one floor 7. Send message to **Elevator Doors Class** to open 8. Start timer 9. Send message to **Elevator Doors Class** to close after timeout 10. Check requests 11. Update requests
COLLABORATION 1. **Elevator Button Class** (subclass) 2. **Floor Button Class** (subclass) 3. **Elevator Doors Class** 4. **Elevator Class**

Figure 11.9 (again)

OOD: Elevator Problem Case Study (contd)

- Responsibilities
  - 8. Start timer
  - 10. Check requests, and
  - 11. Update requests

  are assigned to the `Elevator Controller Class`

- Because they are carried out by the elevator controller

OOD: Elevator Problem Case Study (contd)

- The remaining eight responsibilities have the form
  - "Send a message to another class to tell it do something"
- These should be assigned to that other class
  - Responsibility-driven design
  - Safety considerations
- Methods `openDoors, closeDoors` are assigned to **`Elevator Doors Class`**
- Methods `turnOffButton, turnOnButton` are assigned to **`Floor Button Class`** and **`Elevator Problem Class`**

Detailed Class Diagram: Elevator Problem

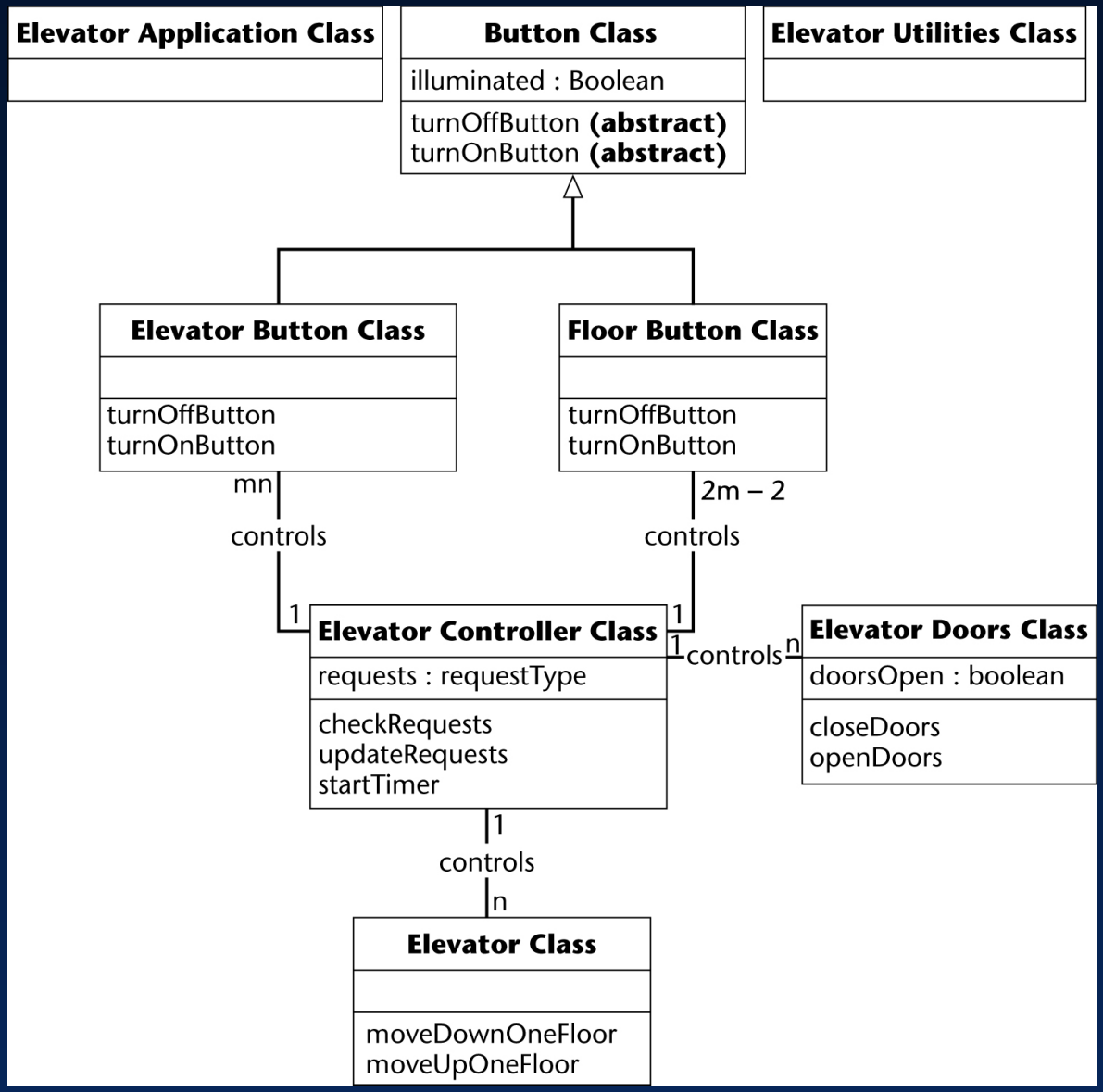

Figure 12.4

Detailed Design: Elevator Problem

- Detailed design of `elevatorEventLoop` is constructed from the statechart

```
void elevatorEventLoop (void)
{
  while (TRUE)
  {
    if (a button has been pressed)
      if (button is not on)
      {
        updateRequests;
        button::turnOnButton;
      }
    else if (elevator is moving up)
    {
      if (there is no request to stop at floor f)
        elevator::moveUpOneFloor;
      else
      {
        stop elevator by not sending a message to move;
        elevatorDoors::openDoors;
        startTimer;
        if (elevatorButton is on)
          elevatorButton::turnOffButton;
        updateRequests;
      }
    }
    else if (elevator is moving down)
      [similar to up case]
    else if (elevator is stopped and request is pending)
    {
      elevatorDoors::closeDoors;
      determine direction of next request;
      if (appropriate floorButton is on)
        floorButton::turnOffButton;
      elevator::moveUp/DownOneFloor;
    }
    else if (elevator is at rest and not (request is pending))
      elevatorDoors::closeDoors;
    else
      there are no requests, elevator is stopped with elevatorDoors closed, so do nothing;
  }
}
```


Figure 12.5

- Step 1. Complete the class diagram

- The final class diagram is shown in the next slide
  - **Date Class** is needed for C++
  - Java has built-it functions for handling dates

Final Class Diagram: MSG Foundation

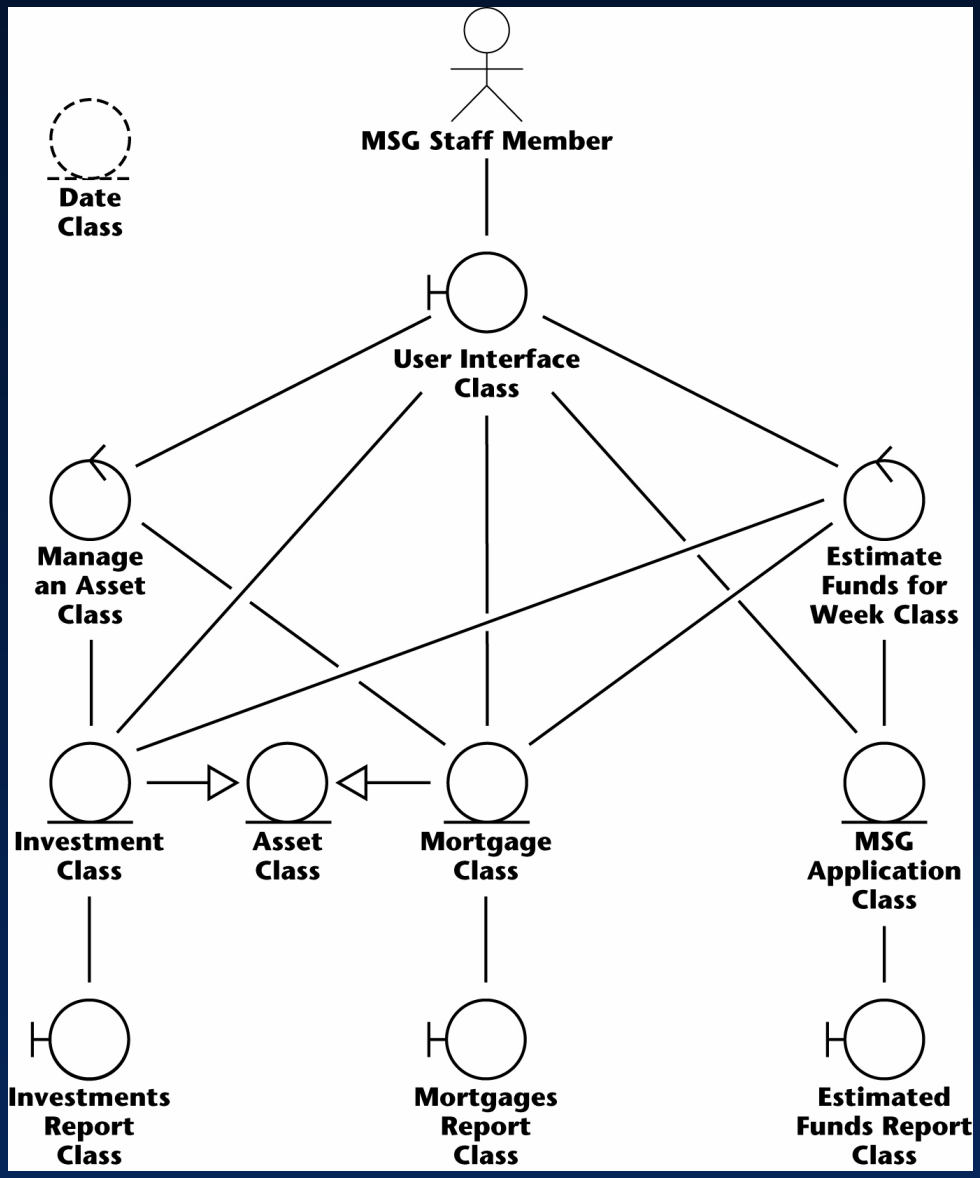

Figure 12.6

Class Diagram with Attribute Formats: MSG Foundation

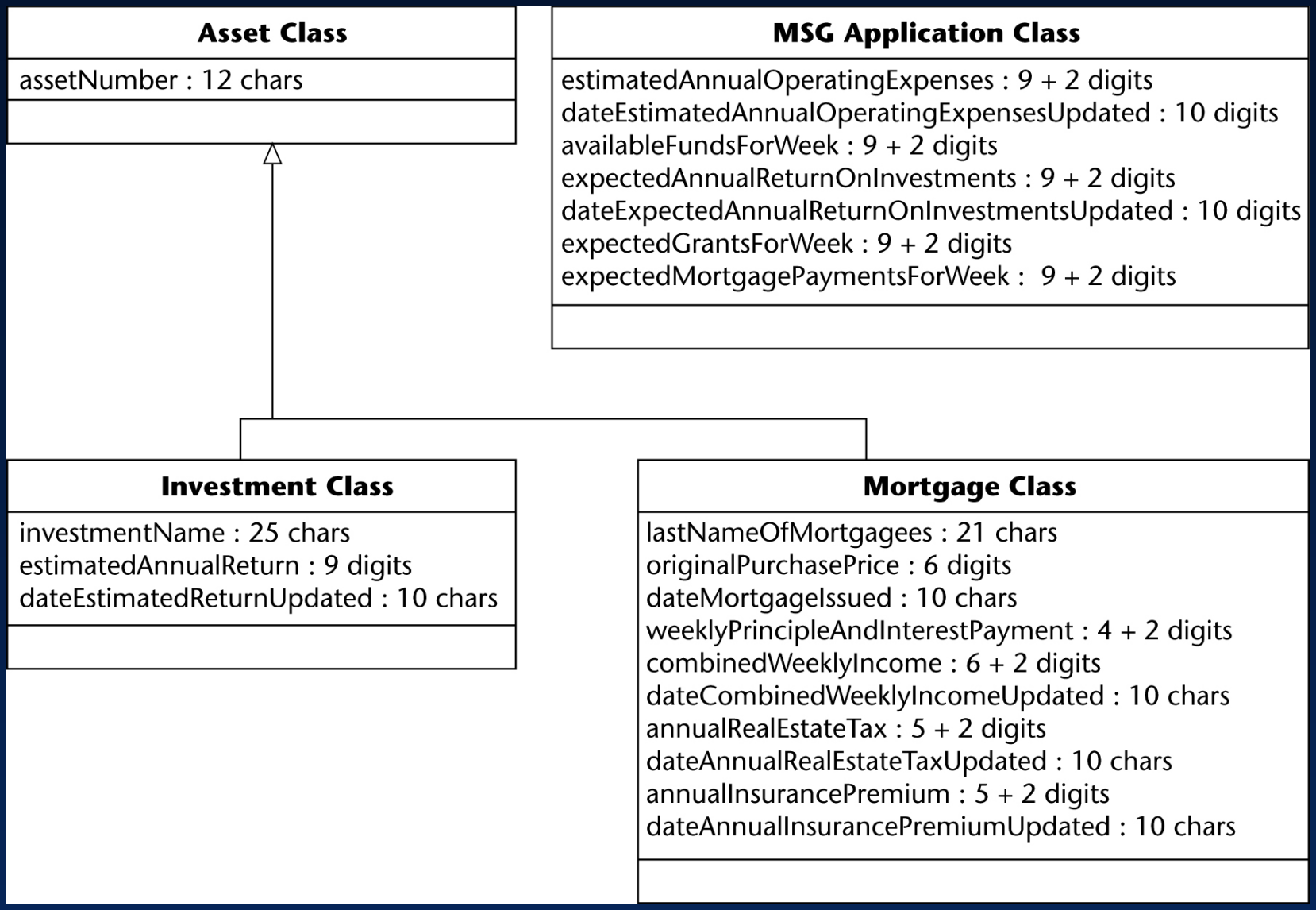

Figure 12.7

Assigning Methods to Classes: MSG Foundation

- Example: `setAssetNumber, getAssetNumber`
  - From the inheritance tree, these accessor/mutator methods should be assigned to **Asset Class**
  - So that they can be inherited by both subclasses of **Asset Class (Investment Class** and **Mortgage Class)**

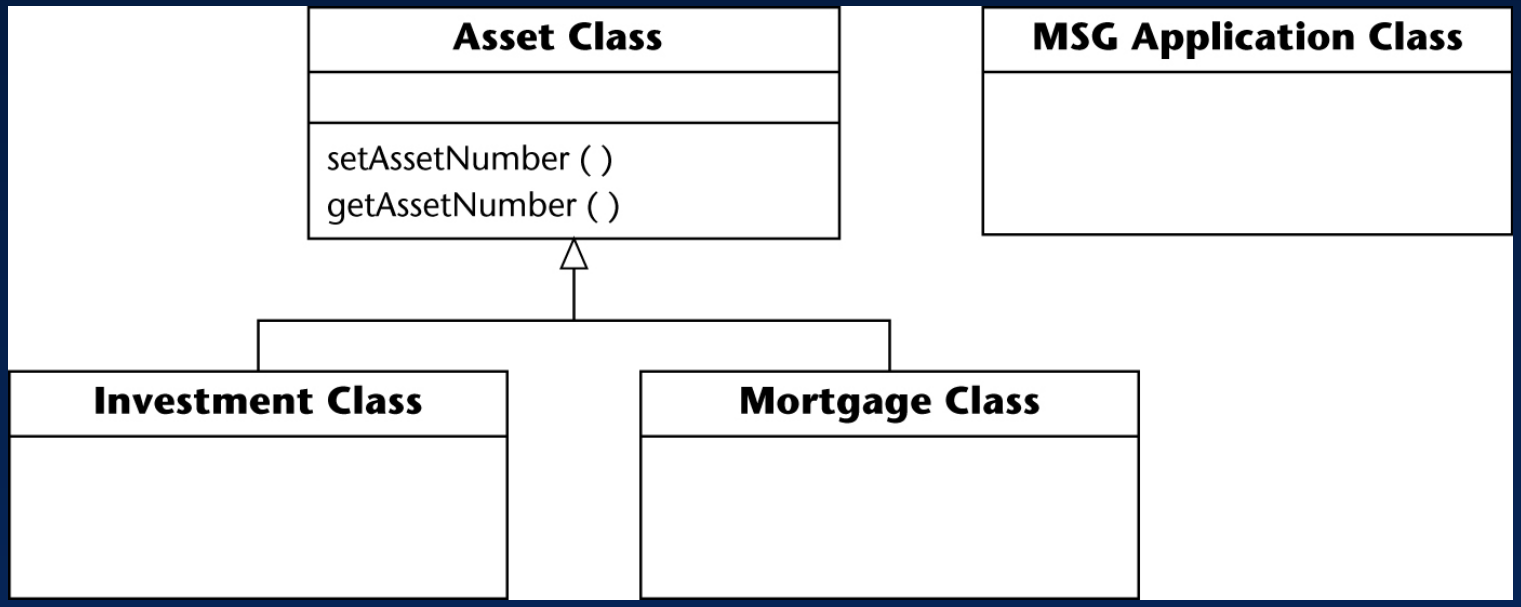

Figure 12.8

Assigning Methods to Classes: MSG Foundation (contd)

- Assigning the other methods is equally straightforward
  - See Appendix F

Detailed Design: MSG Foundation

- Step 2. Perform the detailed design
- Determine what each method does
- Represent the detailed design in an appropriate format
  - Tabular format for method `find` of class **`Mortgage`**
    - » (see Slide 12.11)
  - PDL (pseudocode) for method `computeEstimatedFunds` of class **`EstimatedFundsForWeek`** **and for m**ethod `totalWeeklyNetPayments` of class **`Mortgage`**
    - » (see Slides 12.13, 12.14)

12.4 The Design Workflow

- Summary of the design workflow:
  - The analysis workflow artifacts are iterated and incremented until the programmers can utilize them
- Decisions to be made include:
  - Implementation language
  - Reuse
  - Portability

The Design Workflow (contd)

- The idea of decomposing a large workflow into independent smaller workflows (*packages*) is carried forward to the design workflow
- The objective is to break up the upcoming implementation workflow into manageable pieces
  - *Subsystems*
- It does not make sense to break up the MSG Foundation case study into subsystems — it is too small

The Design Workflow (contd)

- Why the product is broken into subsystems:
  - It is easier to implement a number of smaller subsystems than one large system
  - If the subsystems are independent, they can be implemented by programming teams working in parallel
    - The software product as a whole can then be delivered sooner

The Design Workflow (contd)

- The *architecture* of a software product includes
  - The various components
  - How they fit together
  - The allocation of components to subsystems
- The task of designing the architecture is specialized
  - It is performed by a software *architect*

The Design Workflow (contd)

- The architect needs to make *trade-offs*
  - Every software product must satisfy its functional requirements (the use cases)
  - It also must satisfy its nonfunctional requirements, including
    - Portability, reliability, robustness, maintainability, and security
  - It must do all these things within budget and time constraints
- The architect must assist the client by laying out the trade-offs

The Design Workflow (contd)

- It is usually impossible to satisfy all the requirements, functional and nonfunctional, within the cost and time constraints
  - Some sort of compromises have to be made
- The client has to
  - Relax some of the requirements;
  - Increase the budget; and/or
  - Move the delivery deadline

The Design Workflow (contd)

- The architecture of a software product is critical
  - The requirements workflow can be fixed during the analysis workflow
  - The analysis workflow can be fixed during the design workflow
  - The design workflow can be fixed during the implementation workflow
- But there is no way to recover from a suboptimal architecture
  - The architecture must immediately be redesigned

12.5 The Test Workflow: Design

- Design reviews must be performed
  - The design must correctly reflect the specifications
  - The design itself must be correct
- Transaction-driven inspections
  - Essential for transaction-oriented products
  - However, they are insufficient — specification-driven inspections are also needed

12.6 The Test Workflow: MSG Foundation

- A design inspection must be performed
  - All aspects of the design must be checked
- Even if no faults are found, the design may be changed during the implementation workflow

12.7 Formal Techniques for Detailed Design

- Implementing a complete product and then proving it correct is hard
- However, use of formal techniques during detailed design can help
  - Correctness proving can be applied to module-sized pieces
  - The design should have fewer faults if it is developed in parallel with a correctness proof
  - If the same programmer does the detailed design and implementation
    - The programmer will have a positive attitude to the detailed design
    - This should lead to fewer faults

12.8 Real-Time Design Techniques

- Difficulties associated with real-time systems
  - Inputs come from the real world
    - » Software has no control over the timing of the inputs
  - Frequently implemented on distributed software
    - » Communications implications
    - » Timing issues
  - Problems of synchronization
    - » Race conditions
    - » Deadlock (deadly embrace)

Real-Time Design Techniques (contd)

- The major difficulty in the design of real-time systems
  - Determining whether the timing constraints are met by the design

Real-Time Design Techniques (contd)

- Most real-time design methods are extensions of non-real-time methods to real-time
- We have limited experience in the use of any real-time methods
- The state-of-the-art is not where we would like it to be

12.9 CASE Tools for Design

- It is critical to check that the design artifacts incorporate all aspects of the analysis
  - To handle analysis and design artifacts we therefore need upperCASE tools

- UpperCASE tools
  - Are built around a data dictionary
  - They incorporate a consistency checker, and
  - Screen and report generators
  - Management tools are sometimes included, for
    - Estimating
    - Planning

CASE Tools for Design (contd)

- Data dictionary entries for class `Asset`

Name	Access Specifier	Description	Narrative
Asset	**package private** (default)	Abstract class Attribute: assetNumber Accessors/mutators: getAssetNumber setAssetNumber Virtual methods: read print write find obtainNewData performDeletion Methods: add delete	Abstract superclass of **Investment** and **Mortgage** classes. Comprises the attributes and methods that enable a user to add or delete an asset.
assetNumber	**protected**	12-digit integer	Unique number returned by method getAssetNumber. The first 10 digits contain the asset number itself, the last 2 digits are check digits.
delete	**public**	Method Return type: **void** Input parameter: None Output parameter: None	This method invokes methods obtain-NewData, save, and UserInterface. pressEnter to add a new asset (investment or mortgage).

Figure 12.9

CASE Tools for Design (contd)

- Examples of tools for the design workflow
  - Commercial tools
    - Software through Pictures
    - IBM Rational Rose
    - Together
  - Open-source tool
    - ArgoUML

12.10 Metrics for Design

- Measures of design quality
  - Cohesion
  - Coupling
  - Fault statistics
- Cyclomatic complexity is problematic
  - Data complexity is ignored
  - It is not used much with the object-oriented paradigm

Metrics for Design (contd)

- Metrics have been put forward for the object-oriented paradigm
  - They have been challenged on both theoretical and experimental grounds

12.11 Challenges of the Design Workflow

- The design team should not do too much
  - The detailed design should not become code
- The design team should not do too little
  - It is essential for the design team to produce a complete detailed design

Challenges of the Design Phase (contd)

- We need to “grow” great designers
- Potential great designers must be
  - Identified,
  - Provided with a formal education,
  - Apprenticed to great designers, and
  - Allowed to interact with other designers
- There must be a specific career path for these designers, with appropriate rewards