CS342 Software Engineering

Dr. Ismail Hababeh
German Jordanian University

Lecture 16
DESIGN WORKFLOW

Adapted from Software Engineering, by Dr. Paul E. Young
& slides by Dr. Mohammad Daoud

Object-Oriented Design (OOD)

- Aim of OOD
  - Design the product in terms of the classes extracted during object-oriented analysis (OOA)

Object-Oriented Design Steps

OOD consists of two steps:

1. Complete the class diagram

   a) Determine the formats of the attributes

   b) Assign each method, either to the class itself or to another class that is responsible for the functionality of the method.

2. Perform the detailed design

Object-Oriented Design – Step 1

a) Attribute Format

The formats of the attributes can be directly inferred from the analysis artifacts

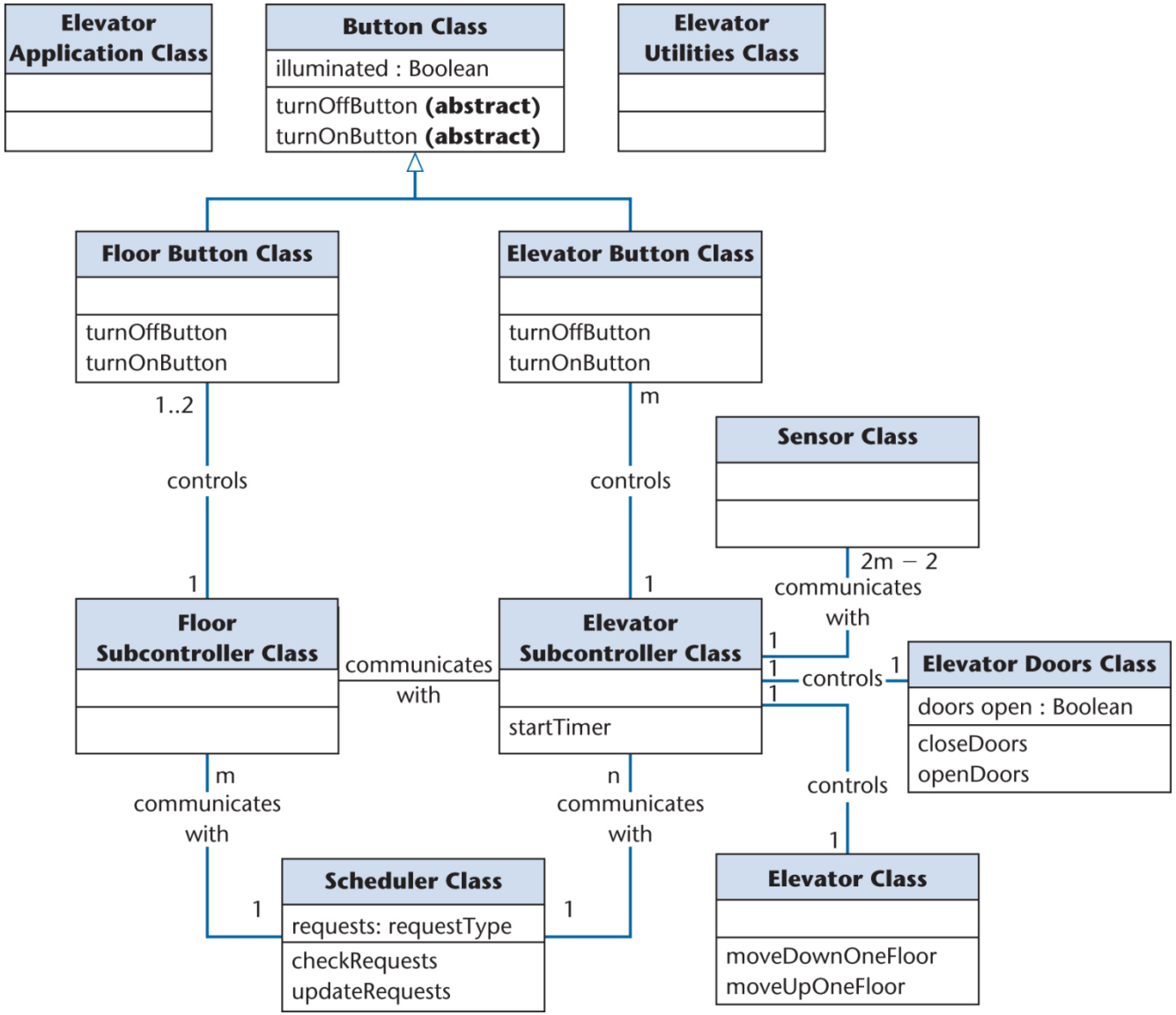
- Example: Dates
  - U.S. format (mm/dd/yyyy)
  - European format (dd/mm/yyyy)
  - In both instances, 10 characters are needed
- The formats could be added during analysis
  - To minimize rework, *never* add an item to a UML diagram until it is strictly necessary

Object-Oriented Design – Step 1

b) Assign Methods to Classes

- Principle 1: Information hiding
- Principle 2: Responsibility-driven design
- Principle 3: If an operation is invoked by many classes, assign the method to the object, not to the class.

Detailed Class Diagram - Elevator



OOD – Detailed Design

- Perform the detailed design
  - A detailed design is developed for all classes and methods
- Technique that can be used
  - Pseudocode
  - Tabular representation

Detailed Class Design - Pseudocode

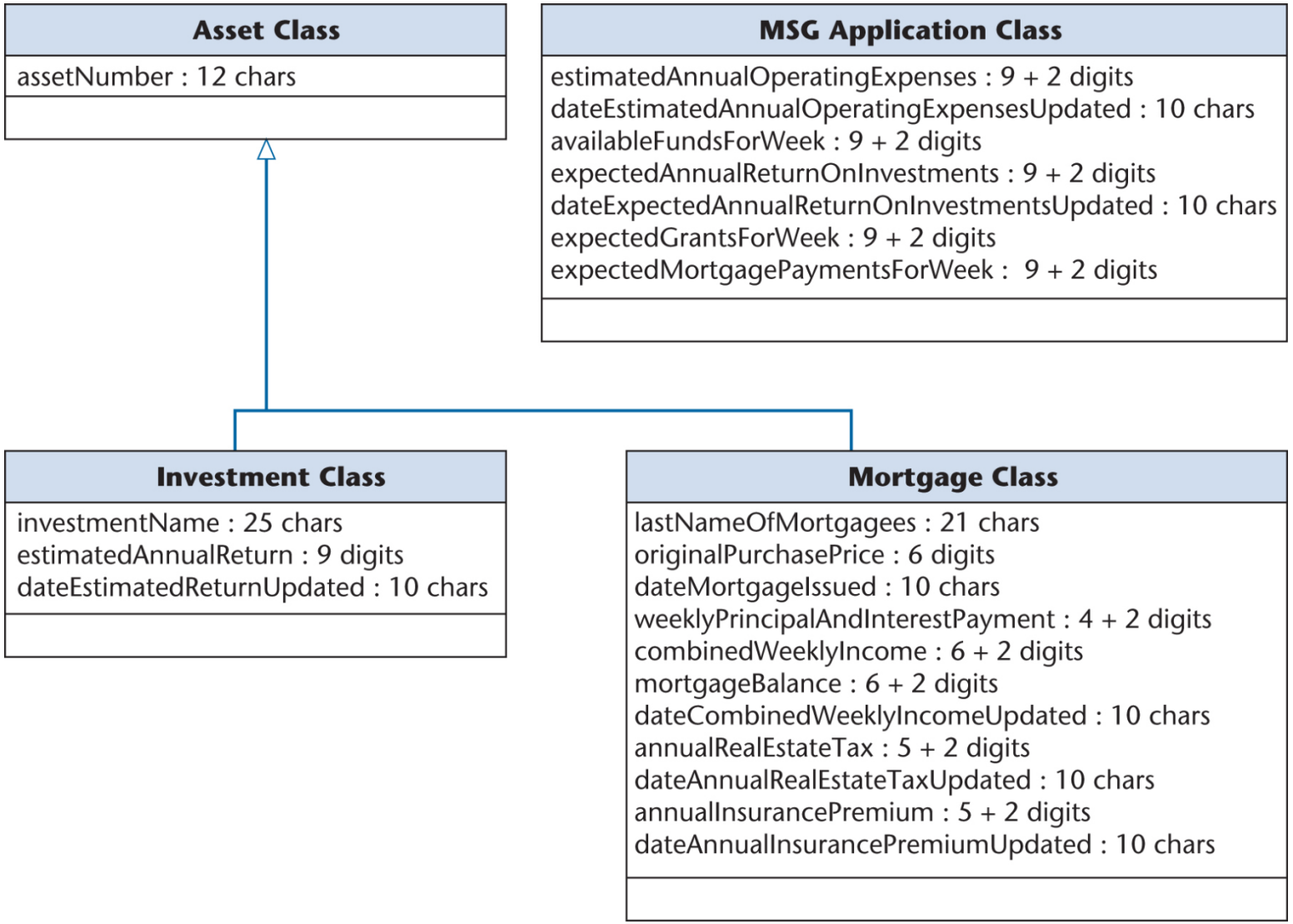
- Part of the Elevator Subcontroller class
- Detailed design of the method `elevatorEventLoop` is constructed

```
void elevatorSubcontrollerEventLoop (void)
{
   while (TRUE)
   {
      if (an elevatorButton has been pressed)
          if (elevatorButton is off)
          {
                  elevatorButton::turnOnButton;
                  scheduler::newRequestMade;
          }
      else if (elevator is moving up)
      {
          wait for sensor message that elevator is arriving at floor;
          scheduler::checkRequests;
          if (there is no request to stop at floor f)
              elevator::moveUpOneFloor;
          else
          {
              stop elevator by not sending a message to move;
              if (elevatorButton is on)
                  elevatorButton::turnOffButton;
              elevatorDoors::openDoors;
              startTimer;
          }
      }
      else if (elevator is moving down)
          [similar to up case]
      else if (elevator is stopped and request is pending)
      {
          wait for timeout;
          elevatorDoors::closeDoors;
          determine direction of next request;
          elevator::moveUp/DownOneFloor;
          wait for sensor message that elevator has left floor;
          floorSubcontroller::elevatorHasLeftFloor;
      }
      else if (elevator is at rest and not (request is pending))
      {
          wait for timeout;
          elevatorDoors::closeDoors;
      }
      else
          there are no requests, elevator is stopped with elevatorDoors closed, so do nothing;
   }
}
```

Object-Oriented Design - MSG Case Study

- Architectural Design
  - Complete the class diagram
    - **Date Class** is needed for C++
    - Java has built-in functions for handling dates
  - Assigning Attributes to Classes
  - Assigning Methods to Classes
- Details Design
  - Determine what each method performs
  - Represent the detailed design in PDL (pseudocode)

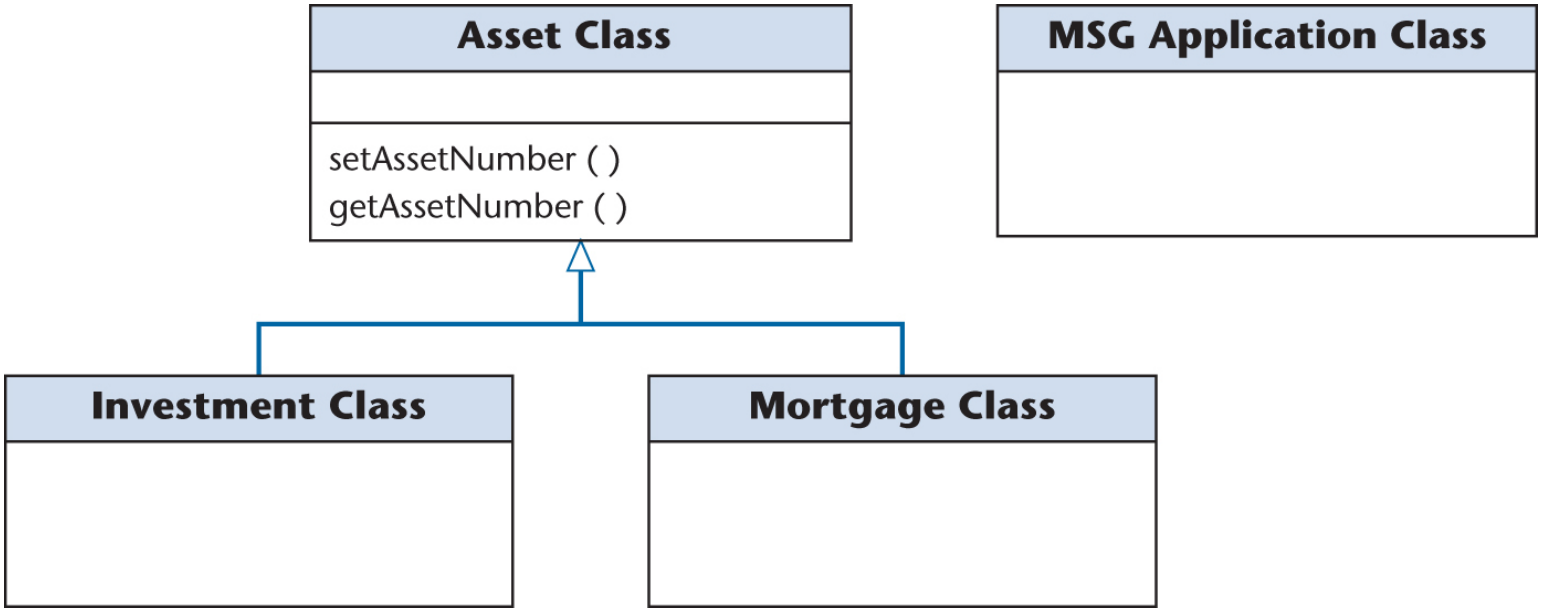
Assigning Attributes to Classes - MSG Case Study



Assigning Methods to Classes - MSG Case Study

- Example: `setAssetNumber, getAssetNumber`
  - From the inheritance tree, these methods should be assigned to **Asset Class**

  So that they can be inherited by both subclasses of **Asset Class (Investment Class** and **Mortgage Class)**



Detailed Design - MSG Case Study

- Determine what each method performes
- Represent the detailed design in an appropriate format
  - PDL (pseudocode)

```
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

```
   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
```

Packages Design

- 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 called *Subsystems.*

Break Down Product into Subsystems

- Why break down product into subsystems?
  - It is easier to implement number of smaller subsystems than one large system
  - If the subsystems are independent, they can be implemented in parallel which in turn minimize the software product implementation time

The Design Architecture

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

The Design Architecture

- The architect needs to make *trade-offs*
  - Every software product must satisfy:
    - Functional requirements (the use cases)
    - Nonfunctional requirements (Portability, reliability, robustness, maintainability, and security)
  - The software product should be performed within budget and time constraints

The Design Architecture

- Usually, software product can't satisfy functional and nonfunctional requirements within the cost and time constraints
  - Some sort of compromises may be made
- The architect should help the client by laying out the trade-offs.
- The client should help the architect by
  - Relax some of the requirements
  - Increase the budget
  - Move the delivery deadline

The Design Architecture Challenge

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

Testing The Design

- Design reviews must be performed
  - The design must correctly reflect the specifications
  - The design itself must be correct

Detailed Design - Formal Techniques

- Implementing a complete product and then proving it correct is hard
- However, use of formal techniques during detailed design can help in
  - Correctness proving can be applied to module-sized pieces
  - The design has 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

Real-Time Design Techniques

- Challenges associated with real-time systems
  - Input comes from the real world ⇨

    software has no control over the timing of the inputs
  - Implemented on distributed software that requires
    - Communications implication
    - Timing issues
  - Problems of synchronization

Real-Time Design Challenges

- The major difficulty in the design of real-time systems is determining that the timing constraints are met by the design
- Most real-time design methods are extensions of non-real-time methods
- Limited experience in the use of real-time methods

Design CASE Tools

- It is critical to check that the design artifacts incorporate all aspects of the analysis
  - To handle analysis and design artifacts we therefore need CASE tools
- Examples of tools for object-oriented design
  - Commercial tools
    - Software through Pictures
    - IBM Rational Rose
    - Together
  - Open-source tool
    - ArgoUML

Design Metrics

- Measures of design quality
  - Cohesion
  - Coupling
  - Fault statistics

Design limitations

- 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 to make it easy for implementation.

Design Professionals

- We need to “grow” great designers
- Potential great designers should be
  - Identified
  - Provided with a formal education
  - Allowed to interact with other designers