OBJECT-ORIENTED ANALYSIS Part I

Object-Oriented and Classical Software Engineering
Sixth Edition, WCB/McGraw-Hill, 2005
Stephen R. Schach
srs@vuse.vanderbilt.edu

Overview

- The analysis workflow
- Extracting the model/entity classes
- Object-oriented analysis: The elevator problem case study
- Functional modeling: The elevator problem case study
- Entity class modeling: The elevator problem case study
- Dynamic modeling: The elevator problem case study
- The test workflow: Object-oriented analysis
- Extracting the boundary and control classes
- The initial functional model: The Osbert Oglesby case study
- The initial class diagram: The Osbert Oglesby case study
- The initial dynamic model: The Osbert Oglesby case study
- Extracting the boundary classes: The Osbert Oglesby case study
- Extracting the boundary classes: The Osbert Oglesby case study

Object-Oriented Analysis

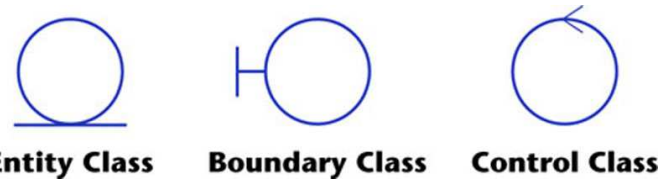
- OOA is a semiformal analysis technique for the object-oriented paradigm
  - There are over 60 equivalent techniques
  - Today, the Unified Process is the only viable alternative
- During this workflow
  - ***The classes are extracted***
- Remark
  - The Unified Process assumes knowledge of class extraction

The Analysis Workflow

- The analysis workflow has two aims
  - Obtain a deeper understanding of the requirements
  - Describe them in a way that will result in a maintainable design and implementation
- There are three types of classes:
  - **Model/Entity classes**: Models **long-lived information**
    - Examples: **Account Class, Painting Class**
  - **Boundary classes**: Models the interaction between the product and the environment. A boundary class is generally **associated with input or output**
    - Examples: **Purchases Report Class, Sales Report Class** (report is one kind of output)
  - **Control classes**: Models complex **computations and algorithms**
    - Examples: **Compute Masterpiece Price Class, Compute Masterwork Price Class, Compute Other Painting Price Class**

UML Notation for These Three Class Types

- Stereotypes (extensions of UML)



Entity Class or Model Class | **Boundary Class** | **Control Class**

Extracting the Model/Entity Classes

- Perform the following three steps incrementally and iteratively
  - Functional modeling
    - Present scenarios of all the use cases (a *scenario* is an instance of a use case)
  - Class modeling
    - Determine the model/entity classes and their attributes
    - Determine the interrelationships and interactions between the model/entity classes
    - Present this information in the form of a ***class diagram***
  - Dynamic modeling
    - Determine the operations performed by or to each model/entity class
    - Present this information in the form of a ***statechart***

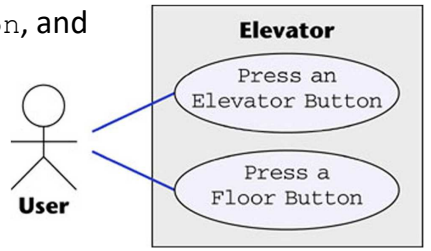
Object-Oriented Analysis: The Elevator Problem Case Study

A product is to be installed to **control n elevators in a building with m floors**. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each **elevator has a set of m buttons**, one for each floor. These illuminate when pressed and **cause the elevator to visit the corresponding floor**. The illumination **is canceled when** the corresponding **floor is visited by the elevator**
2. **Each floor**, *except the first and the top floor*, *has two buttons*, one to request an **up-elevator**, one to request a **down-elevator**. These buttons illuminate when pressed. The illumination **is canceled when an elevator visits the floor**, then moves in the desired direction
3. **If an elevator has no requests**, **it remains at its current floor with its doors closed**

Functional Modeling: The Elevator Problem Case Study

- A use case describes the interaction between
  - The product, and
  - The actors (external users)
- For the elevator problem, there are only two possible use cases
  - `Press an Elevator Button`, and
  - `Press a Floor Button`



Scenarios

- A use case provides a generic description of the overall functionality
- A scenario (สถานการณ์ที่นำเสนอในรูปของ step flow) is an instance of a use case
- Sufficient scenarios need to be studied to get a comprehensive insight into the target product being modeled

Normal Scenario: Elevator Problem

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator doors open.
6. The timer starts.
   User A enters the elevator.
7. User A presses the elevator button for floor 7.
8. The elevator button for floor 7 is turned on.
9. The elevator doors close after a timeout.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.
    User A exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 9 with User B.

Exception Scenario: Elevator Problem

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator doors open.
6. The timer starts.
   User A enters the elevator.
7. User A presses the elevator button for floor 1.
8. The elevator button for floor 1 is turned on.
9. The elevator doors close after a timeout.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
    User B exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 1 with User A.

Entity Class Modeling : The Elevator Problem Case Study

- Extract classes and their attributes
  - Represent them using a UML diagram
- One alternative: Deduce (อนุมาน) the classes from use cases and their scenarios
  - Possible danger: Often there are many scenarios, and hence
  - Too many candidate classes
- Other alternatives:
  - CRC cards (if you have domain knowledge)
  - Noun extraction

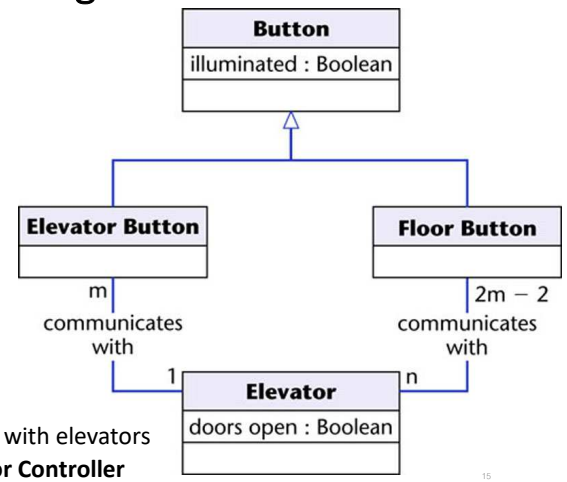
Noun Extraction

- **A two-stage process**
- **Stage 1.** Concise problem definition
  - Describe the software product in single paragraph
  - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors.
  - Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied.
  - When an elevator has no requests, it remains at its current floor with its doors closed
- **Stage 2.** Identify the nouns
  - Identify the nouns in the informal strategy
  - Buttons in elevators and on the floors control the movement of n elevators in a building with m floors. Buttons illuminate when pressed to request the elevator to stop at a specific floor; the illumination is canceled when the request has been satisfied. When an elevator has no requests, it remains at its current floor with its doors closed
- **Use the nouns as candidate classes**

Noun Extraction (contd)

- Nouns
  - button, elevator, floor, movement, building, illumination, request, door
  - floor, building, door are outside the problem boundary — exclude
  - movement, illumination, request are abstract nouns — exclude (they may become attributes)
- Candidate classes:
  - **Elevator** and **Button**
- Subclasses:
  - **Elevator Button** and **Floor Button**

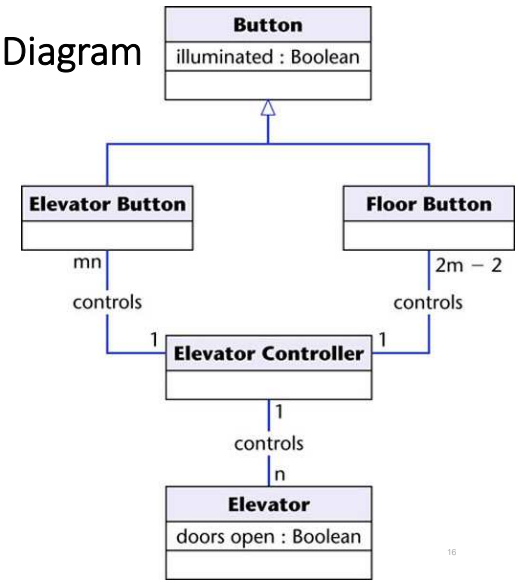
First Iteration of Class Diagram



- Problem
  - Buttons do not communicate directly with elevators
  - We need an additional class: **Elevator Controller**

Second Iteration of Class Diagram

- All relationships are now 1-to-n
  - This makes design and implementation easier



CRC Cards

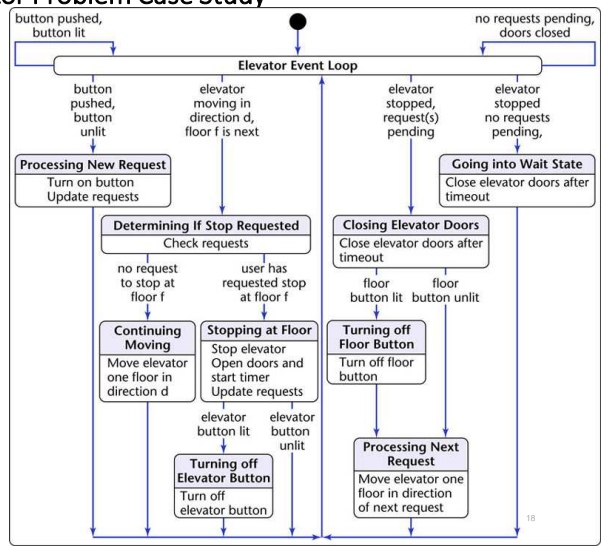
- Used since 1989 for OOA
- For each class, fill in a card showing
  - Name of Class
  - Functionality (Responsibility)
  - List of classes it invokes (Collaboration)
- Now CRC cards are automated (CASE tool component)

- Strength
  - When acted out by team members, CRC cards are a powerful tool for highlighting missing or incorrect items
- Weakness
  - If CRC cards are used to identify model/entity classes, domain expertise is needed

Dynamic Modeling: The Elevator Problem Case Study

- Produce a UML statechart
- State, event, and predicate are distributed over the statechart

- This is shown by considering specific scenarios
- In fact, a statechart is constructed by modeling the events of the scenarios



The Test Workflow: Object-Oriented Analysis

- CRC cards are an excellent testing technique

CLASS **Elevator Controller**
RESPONSIBILITY 1. Turn on elevator button 2. Turn off elevator button 3. Turn on floor button 4. Turn off floor button 5. Move elevator up one floor 6. Move elevator down one floor 7. Open elevator doors and start timer 8. Close elevator doors after timeout 9. Check requests 10. Update requests
COLLABORATION 1. Class **Elevator Button** 2. Class **Floor Button** 3. Class **Elevator**

CRC Cards

- Consider responsibility
  - 1. Turn on elevator button
- This is totally inappropriate for the object-oriented paradigm
  - Responsibility-driven design has been ignored
  - Information hiding has been ignored
- Responsibility
  - 1. Turn on elevator button

  should be

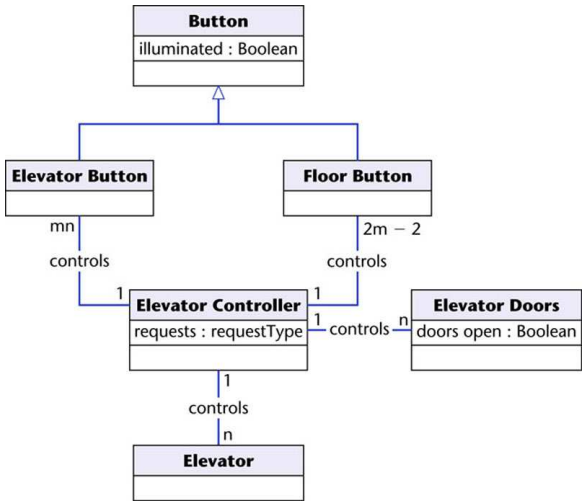
  1. Send message to Elevator Button to turn itself on
- Also, a class has been overlooked (มองข้าม)
- The elevator doors have a *state* that changes during execution (class characteristic)
  - Add class **Elevator Doors**
  - Safety considerations
- Modify the CRC card

Second Iteration of the CRC Card

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

- Having modified the class diagram, reconsider the
  - Use-case diagram (no change)
  - Class diagram (see the next slide)
  - Statecharts
  - Scenarios (see the slide after the next slide)

Third Iteration of Class Diagram



Second Iteration of the Normal Scenario:

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The floor button informs the elevator controller that the floor button has been pushed.
3. The elevator controller sends a message to the Up floor button to turn itself on.
4. The elevator controller sends a series of messages to the elevator to move itself up to floor 3. The elevator contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
5. The elevator controller sends a message to the Up floor button to turn itself off.
6. The elevator controller sends a message to the elevator doors to open themselves.
7. The elevator control starts the timer.
   User A enters the elevator.
8. User A presses elevator button for floor 7.
9. The elevator button informs the elevator controller that the elevator button has been pushed.
10. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
11. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
12. The elevator controller sends a series of messages to the elevator to move itself up to floor 7.
13. The elevator controller sends a message to the elevator button for floor 7 to turn itself off.
14. The elevator controller sends a message to the elevator doors to open themselves to allow User A to exit from the elevator.
15. The elevator controller starts the timer.
    User A exits from the elevator.
16. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
17. The elevator controller sends a series of messages to the elevator to move itself up to floor 9 with User B.

- The object-oriented analysis is now fine
- We should rather say:
  - The object-oriented analysis is fine *for now*
- We may need to return to the object-oriented analysis workflow during the object-oriented design workflow

เพิ่ม send a message
-> เนื่องจาก encapsulation
และแบ่งแยกหน้าที่ตาม MVC model

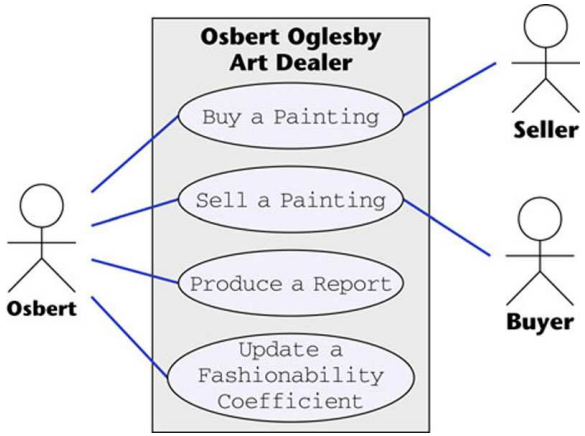
Extracting the Boundary and Control Classes

- Each
  - Input screen,
  - Output screen, and
  - Report

  is modeled by its own boundary class
- Each nontrivial computation (การคำนวณที่ซับซ้อน) is modeled by a control class

The Initial Functional Model: Osbert Oglesby Case Study

- Recall the Osbert Oglesby use-case diagram:



First Scenario of Use Case Buy a Masterpiece

- Normal scenario

Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work.

   Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).

- Only four of the six paragraphs in the scenario are numbered
  - The two unnumbered sentences
    - "Osbert wishes to buy a masterpiece" and
    - "Osbert makes an offer below the maximum purchase price — the offer is accepted by the seller"

    have nothing to do with the interaction between Osbert and the software product
- These unnumbered paragraphs are essentially comments

Second Scenario of Use Case `Buy a Masterpiece`

- Exception scenario

Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work.

   Osbert makes an offer below the maximum purchase price. The seller turns down Osbert's offer.

Third Scenario of Use Case `Buy a Masterpiece`

- This is another exception scenario

Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product reports that there are no similar works.

   Osbert does not make an offer for the painting.

Extended Scenario of Use Case `Buy a Masterpiece`

- Normal and exception scenarios can be combined into an extended scenario

Osbert Oglesby wishes to buy a masterpiece.
1. Osbert enters the description of the painting.
2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist.
3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work.

   Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller.
4. Osbert enters sales information (name and address of seller, purchase price).

Possible alternatives:

A. The seller turns down Osbert's offer.
B. No similar painting by that artist is in the auction file, so Osbert does not make an offer for the painting.

The Initial Class Diagram: *The Osbert Oglesby Case Study*

- The aim of entity modeling step is to extract the model/entity classes, determine their interrelationships, and find their attributes
- Usually, the best way to begin this step is to use the two-stage noun extraction method

- **Noun Extraction**
- **Stage 1:** Describe the software product in one paragraph:
  - Reports are to be generated in order to improve the effectiveness of the decision-making process for buying works of art.
  - The reports contain buying and selling information about paintings, which are classified as masterpieces, masterworks, and other paintings
- **Stage 2:** Identify the nouns in this paragraph
  - Reports are to be generated in order to improve the effectiveness of the decision-making process for buying works of art.
  - The reports contain buying and selling information about paintings, which are classified as masterpieces, masterworks, and other paintings

Noun Extraction: Osbert Oglesby (contd)

- The nouns are report, effectiveness, process, buying, work of art, selling, information, painting, masterpiece, and masterwork
- effectiveness, process and information are abstract nouns and are therefore unlikely to be entity classes
- Nouns buying and selling are derived from the verbs "buy" and "sell"
  - They will probably be operations of some class
- Noun report is much more likely to be a boundary class than an entity class
- Noun work of art is just a synonym for painting

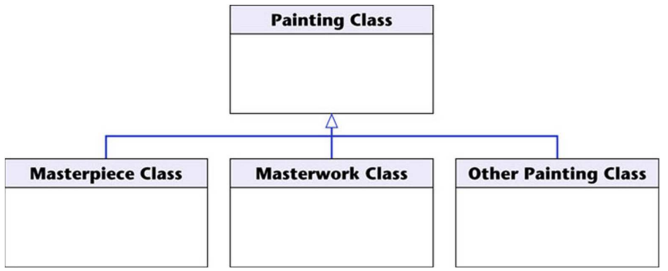
First Iteration of the Initial Class Diagram

- This leaves four candidate entity classes:
  - **Painting Class, Masterpiece Class, Masterwork Class,** and **Other Painting Class**

Osbert Oglesby wishes to buy a masterpiece. 1. Osbert enters the description of the painting. 2. The software product scans the auction records to find the price and year of the sale of the most similar work by the same artist. 3. The software product computes the maximum purchase price by adding 8.5%, compounded annually, for each year since the auction of the most similar work. Osbert makes an offer below the maximum purchase price—the offer is accepted by the seller. 4. Osbert enters sales information (name and address of seller, purchase price).
Possible alternatives: A. The seller turns down Osbert's offer. B. No similar painting by that artist is in the auction file, so Osbert does not make an offer for the painting.

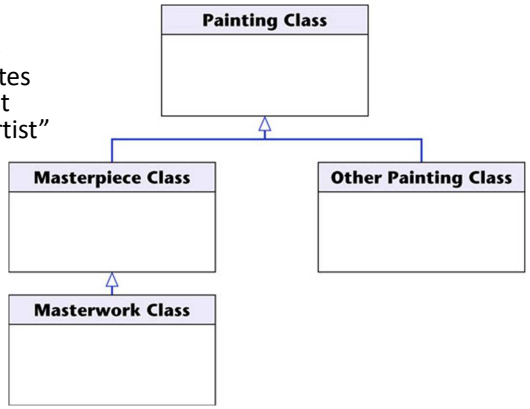
Second Iteration of the Initial Class Diagram

- Consider the interrelationships between the entity classes
- A masterpiece is a specific type of painting, and so is a masterwork and an "other painting"
  - **Painting Class** is therefore the base class
  - **Masterpiece Class**, **Masterwork Class,** and **Other Painting Class** are subclasses of that base class



Third Iteration of the Initial Class Diagram

- The class diagram does not reflect aspects of the pricing algorithm
- When dealing with a masterwork
  - "The software product first computes the maximum purchase price as if it were a masterpiece by the same artist"
- That is, a masterwork has to have all the attributes of a masterpiece (so that its maximum purchase price can be computed as if it were a masterpiece) and, in addition, it may have attributes of its own

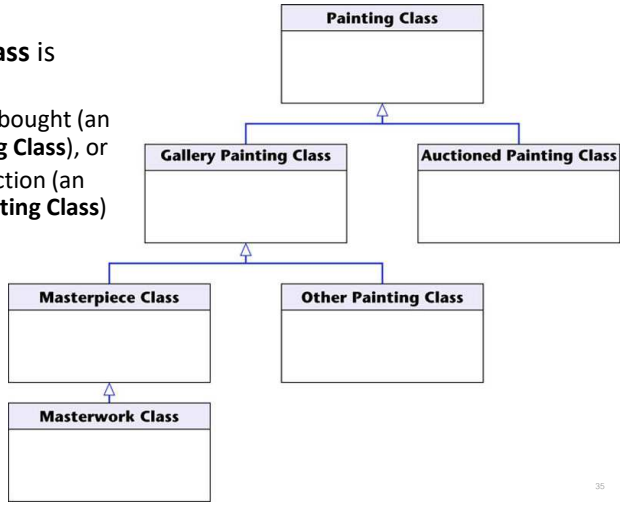


Fourth Iteration of the Initial Class Diagram

- Another aspect of the pricing algorithm that is not reflected in the current class diagram is
  - "The software product computes the coefficient of similarity between each painting for which there is an auction record and the painting under consideration for purchase"
- **Auctioned Painting Class** is needed to make these comparisons
  - An auctioned painting must be a subclass of **Painting Class**
  - But a painting previously been sold at an auction somewhere in the world has nothing to do with paintings currently on display for sale in Osbert's gallery

Fourth Iteration of Initial Class Diagram (contd)

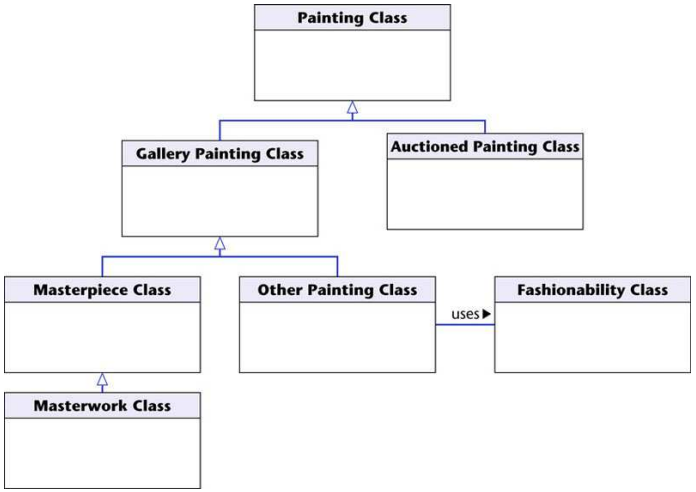
- An instance of **Painting Class** is either
  - A painting that Osbert has bought (an instance of **Gallery Painting Class**), or
  - A painting sold at some auction (an instance of **Auctioned Painting Class**)



Fifth Iteration of the Initial Class Diagram

- A third aspect of the maximum price algorithm that has not yet been modeled is *fashionability*
  - "The software product computes the maximum purchase price from the formula $F \times A$, where F is a constant for that artist (fashionability coefficient) …"
- **Fashionability Class** is needed
  - A painting of **Other Painting Class** can then use the instance of **Fashionability Class** for that artist to compute the maximum price that Osbert should offer to pay

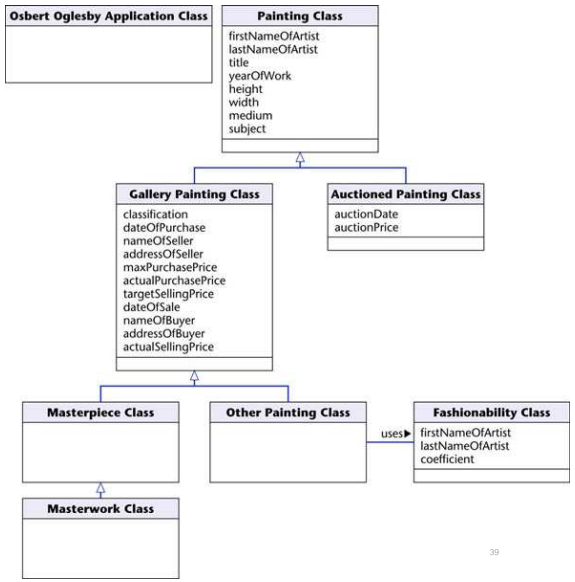
Fifth Iteration of the Initial Class Diagram (contd)



Initial Class Diagram:Osbert Oglesby (contd)

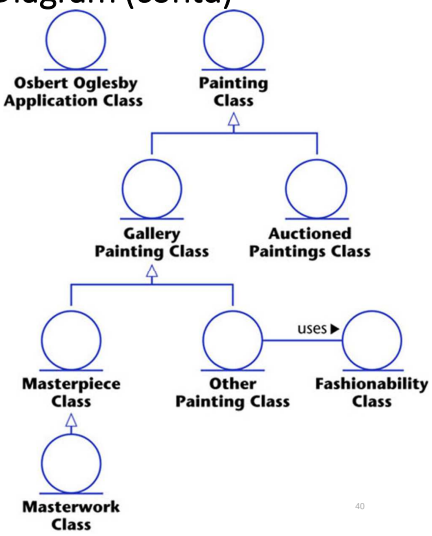
- Why was the first iteration of the class diagram so inadequate?
  - The Osbert Oglesby case study appears to be a straightforward data-processing application
  - The one-paragraph description correctly did not incorporate the pricing algorithm
- Unfortunately, the algorithmic details turned out to be critical to the class diagram
- The first iteration of the class diagram was no good
  - However, repeated iteration and increment led to a reasonable class diagram
- This demonstrates the power of the iterative and incremental approach
- Finally, we add the attributes of each class to the class diagram
  - For the Osbert Oglesby case study, the result is shown on the next slide
- The empty rectangle at the bottom of each box will later be filled with the operations of that class

Fifth Iteration of the Initial Class Diagram (contd)



Fifth Iteration of the Initial Class Diagram (contd)

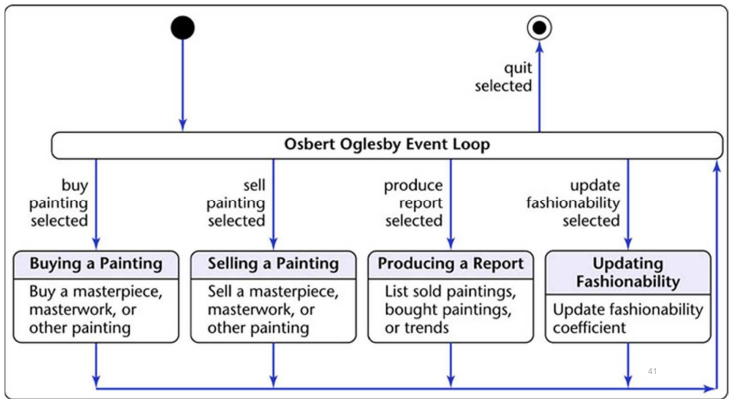
- **Osbert Oglesby Application Class** will contain the operation that starts execution of the whole software product
- The next slide shows the fifth iteration of the initial class diagram, without the attributes, but explicitly reflecting the stereotypes
  - All eight classes in that figure are entity classes
- This is also a class diagram
  - **A** class diagram depicts classes and their interrelationships; attributes and operations are optional



The Initial Dynamic Model: The Osbert Oglesby Case Study

- Dynamic modeling is the third step in extracting the entity classes
- A statechart is constructed that reflects all the operations performed by or to the software product
- The operations are determined from the scenarios

Initial statechart



41

Initial Dynamic Model: Osbert Oglesby (contd)

- The solid circle (top left) represents the initial state
- The white circle with the small black circle inside (top right) represents the final state
- States other than the initial and final states are represented by rectangles with rounded corners
- The arrows represent transitions from state to state

- In state **Osbert Oglesby Event Loop**, one of five events can occur:
  - buy painting selected
  - sell painting selected
  - print report selected
  - update fashionability selected
  - quit selected

42

Initial Main Menu: Osbert Oglesby

- Graphical user interface (GUI)
  - "Point and click"

- **Dynamic Modeling**
- In the object-oriented paradigm, there is a dynamic model for each class, rather than for the system as a whole, as in this case study
  - However, objects in this software product never move from one class to another class

- Accordingly, a dynamic model for the software product as a whole is appropriate

43

Extracting the Boundary Classes: The Osbert Oglesby Case Study

- It is usually easy to extract boundary classes
  - Each input screen, output screen, and printed report is generally modeled by a boundary class
- One screen should be adequate for all four Osbert Oglesby use cases
- Thus there is one initial boundary class
  - **User Interface Class**

- A GUI needs special software
  - However, a textual interface runs on all computers

```
MAIN MENU
OSBERT OGLESBY – ART DEALER
    1. Buy a painting
    2. Sell a painting
    3. Produce a report
    4. Update fashionability
    5. Quit
Type your choice and press <ENTER>:
```

44

Initial Boundary Classes: Osbert Oglesby (contd)

- There are three reports:
  - The purchases report
  - The sales report
  - The future trends report
- The content of each report is different
  - Each report therefore has to be modeled by a separate boundary class
- There are therefore four initial boundary classes

```
User Interface Class
Purchases Report Class
Sales Report Class
Future Trends Report Class
```