객체지향설계
Designing with Objects
(객체를 사용한 설계)

충남대학교
컴퓨터공학과
이만호

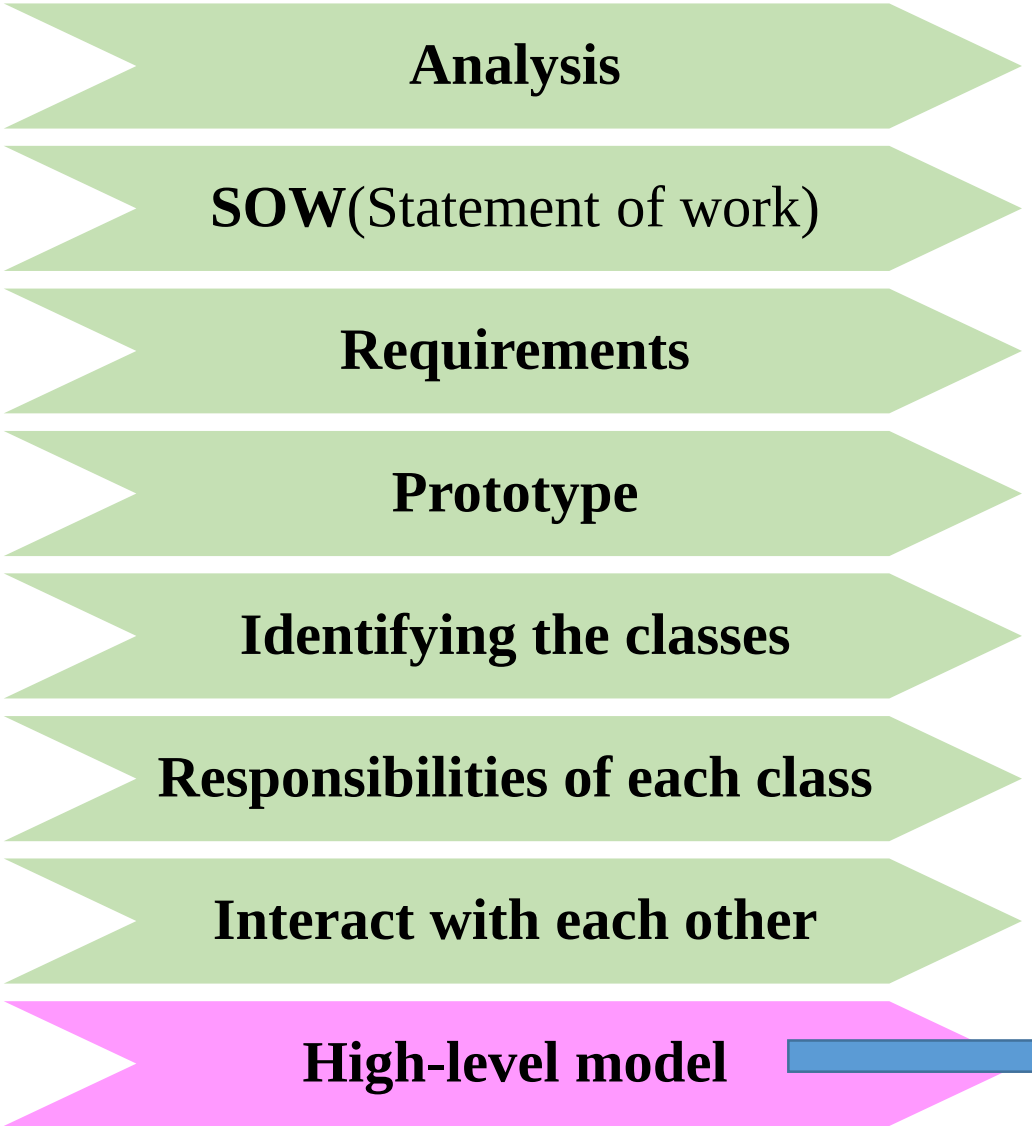
학습 내용

- Design Guidelines
- Case Study : A Blackjack Example

Design Guidelines

- Design Guidelines
  - Many design methodologies available today.
    - The primary issue is not which design method to use, but simply whether to use a method at all.
  - Many organizations do not follow a standard software development process.
    - The most important factor in creating a good design is to find a process that you and your organization can feel comfortable with.
- Generally, a solid OO design process includes the following steps:
  1. Doing the proper analysis
  2. Developing a statement of work (SOW) that describes the system
  3. Gathering the requirements from this statement of work
  4. Developing a prototype for the user interface
  5. Identifying the classes
  6. Determining the responsibilities of each class
  7. Determining how the various classes interact with each other
  8. Creating a high-level model that describes the system → UML(Unified Modeling Language)

OO Design Process



- The system, or object model, is made up of class diagrams and class interactions.
- This model should represent the system faithfully and be easy to understand and modify.
- We also need a notation for the model.
- UML (Unified Modeling Language) is a good modeling tool. (not a design process)

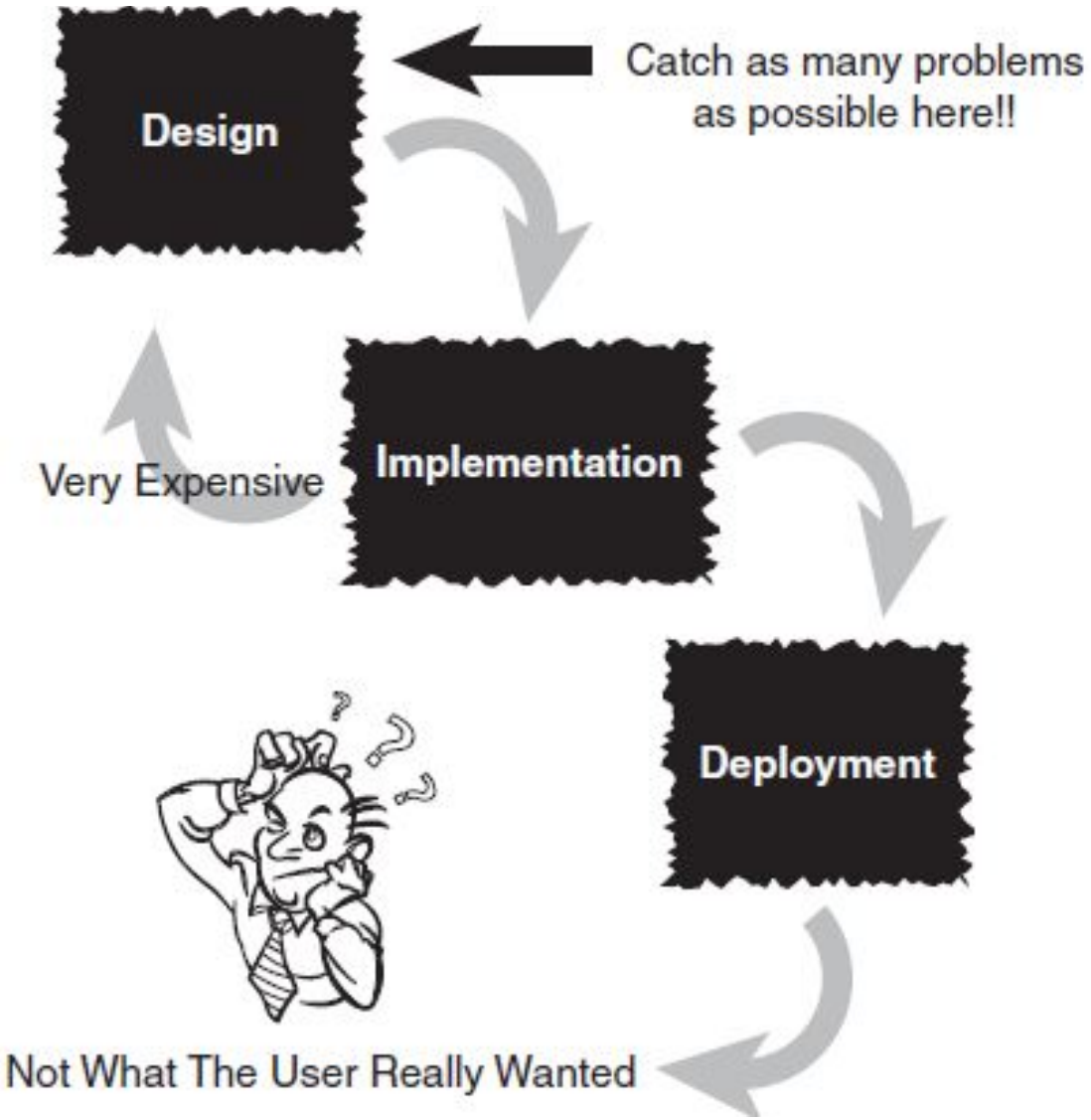
The Ongoing Design Process(지속적인 설계 과정)

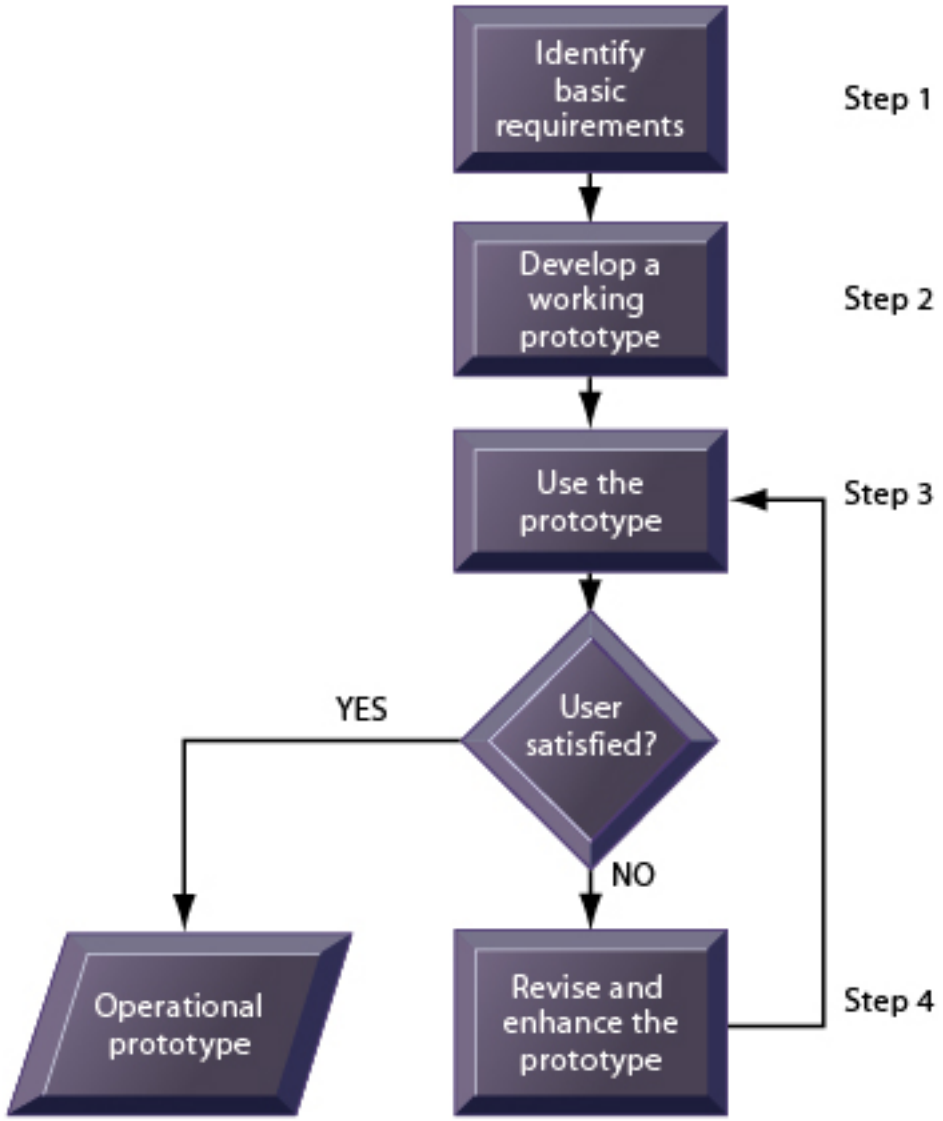
- Design is an ongoing process.
  - Despite the best intentions and planning, in all but the most trivial cases.
  - Even after a product is in testing, design changes will pop up.
  - It is up to the project manager to draw the line that says when to stop changing a product and adding features.

Design Methodologies

- Waterfall Model
  - Early methodology
  - Advocates (recommends) strict boundaries between the various phases.
  - In practice, the waterfall model has been found to be unrealistic.
  - Despite the recent aversion to the waterfall model, the goal behind the model is understandable. (see the figure of the next slide)
    - Coming up with a complete and thorough design before starting to code is a sound practice.
    - Iterating across phase boundaries is unavoidable; however, you should keep these iterations to a minimum.
- Others
  - Rapid prototyping model is one of them. (see the figure of the next page)
  - Promote a true iterative process.
  - Some implementation is attempted prior to completing the design phase as a type of proof-of-concept (POC).

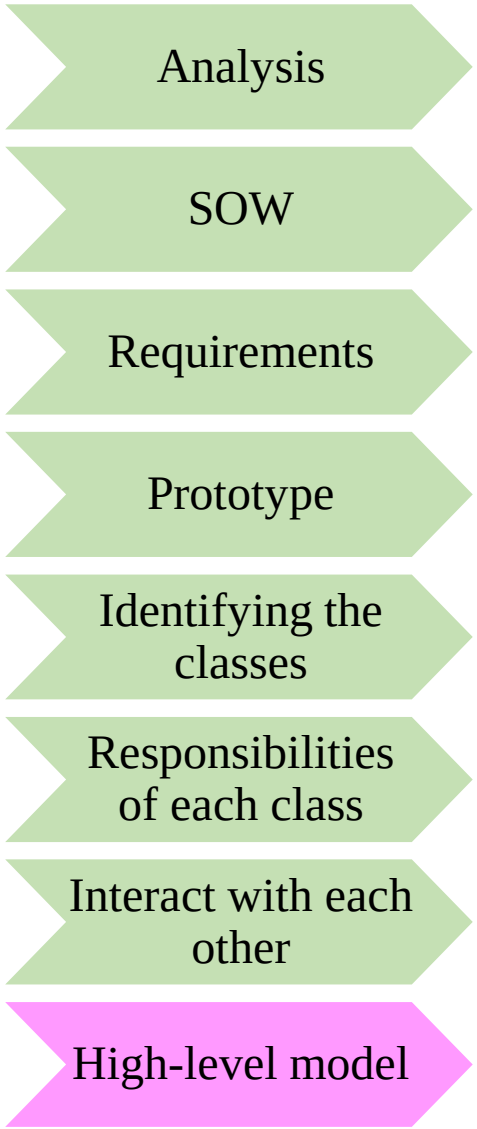
Waterfall Model & Rapid Prototyping



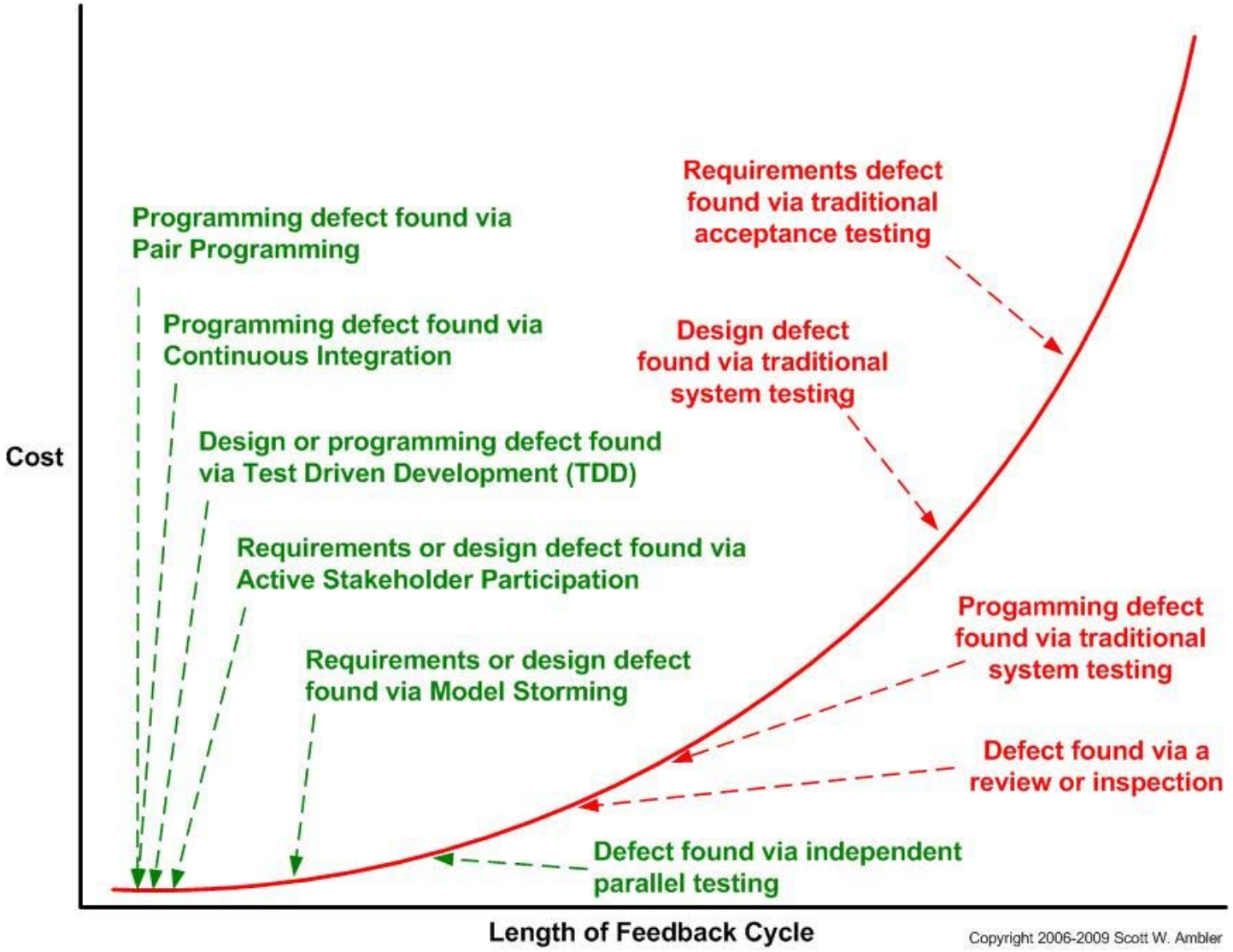


OO Design Process – Requirements

- The reasons to identify requirements early and keep design changes to a minimum are as follows:
  - The cost of a design change in the design phase is relatively small.
  - The cost of a design change in the implementation phase is significantly higher.
  - The cost of a design change after the deployment phase is astronomical when compared to the first item.



Cost of Change Curve

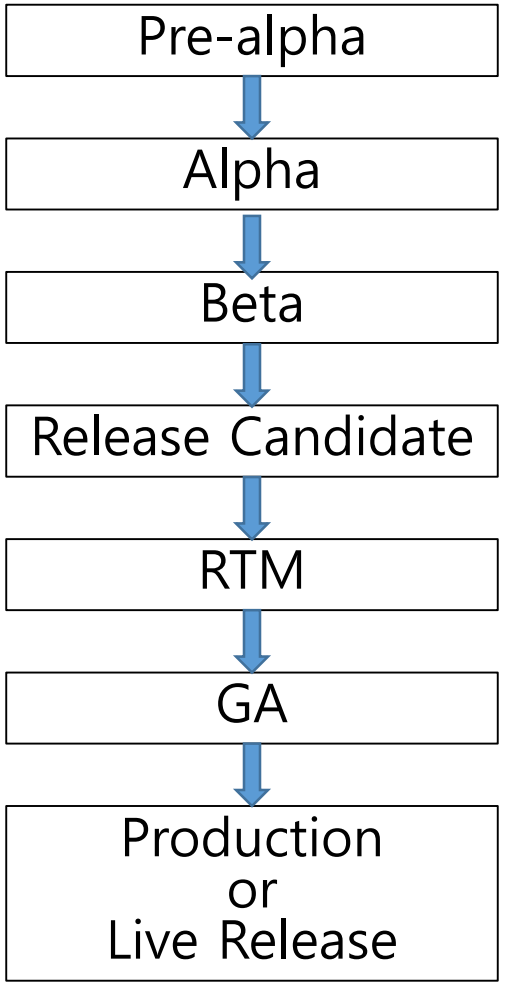


Safety vs. Economics

- Crossing a bridge that has not been inspected and tested.
  - The construction company can save money.
  - Passengers are tester.
  - The bridge may be collapsed.
- With many software packages, users are left with the responsibility of doing much of the testing.
  - This is very costly for both the users and the software providers.
  - Unfortunately, short-term economics often seem to be the primary factor in making project decisions.

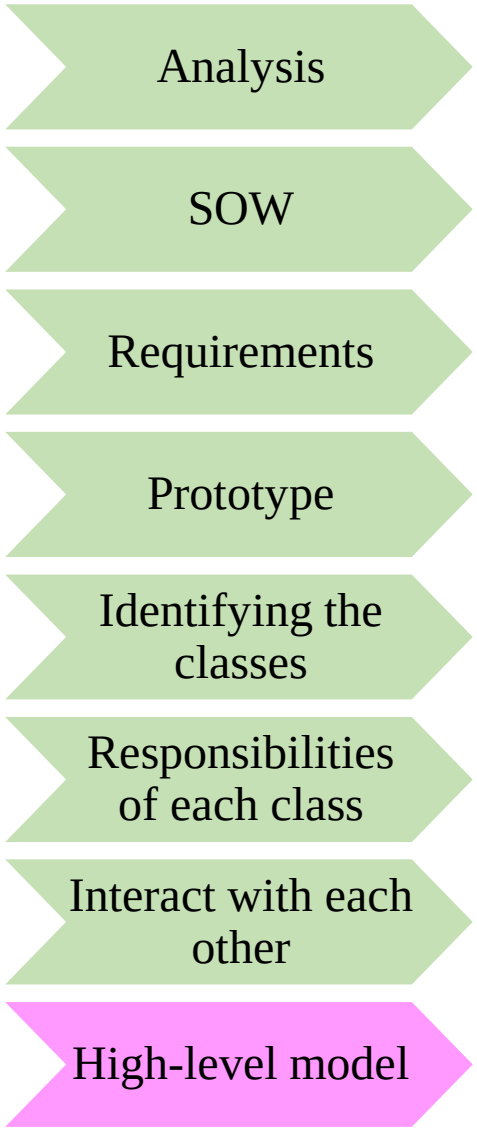
Software Release Life Cycle

1. Stages of development
    1. Pre-alpha
    2. Alpha
    3. Beta
    4. Release candidate
2. Release
    1. Release to manufacturing (RTM)
    2. General availability (GA): marketing stage
3. EOL (End-of-Life)
    1. EOLA (end of life announcement)
    2. LOD (last order date)
    3. EOL (end-of-life)



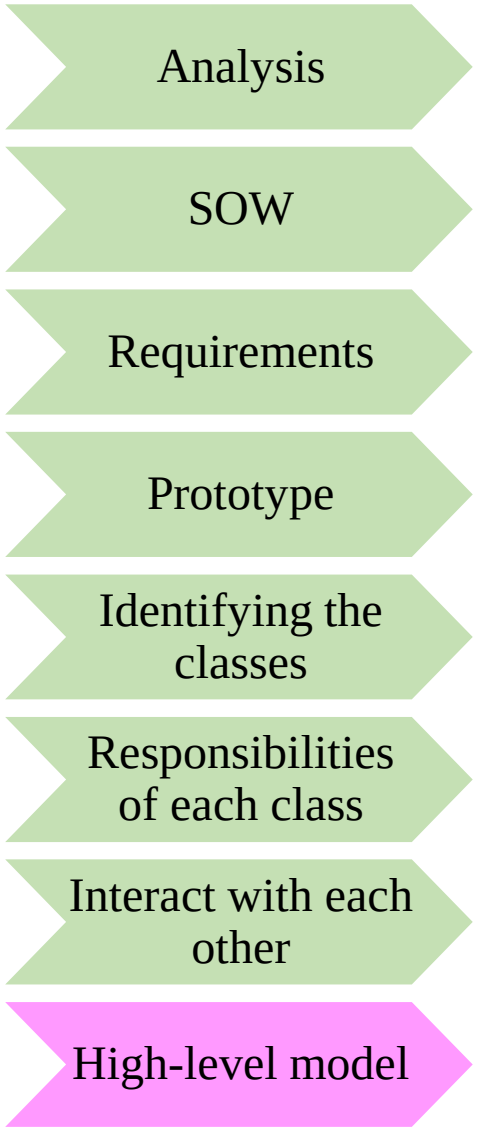
OO Design Process – Analysis

- Performing the Proper Analysis
  - The users and the developers must do the proper research and analysis to determine the followings:
    - The statement of work (SOW)
    - The requirements of the project
    - Whether to actually do the project.
      - There must not be any hesitation to terminate the project if there is a valid reason to do so.
- The analysis phase is not specific to OO
  - They apply to software development in general.



OO Design Process – SOW

- Developing a SOW(Statement of Work)
  - The SOW is a document that describes the system.
  - The SOW contains everything that must be known about the system.
  - Many customers create a request for proposal (RFP) for distribution, which is similar to the statement of work.
- RFP(request for proposal)
  - Completely describes the system they want built, and releases it to multiple vendors.
  - The vendors use the RFP, along with whatever analysis they need to do, to determine whether they should bid on the project, and if so, what price to charge.



Standard Checklist of SOW – 1

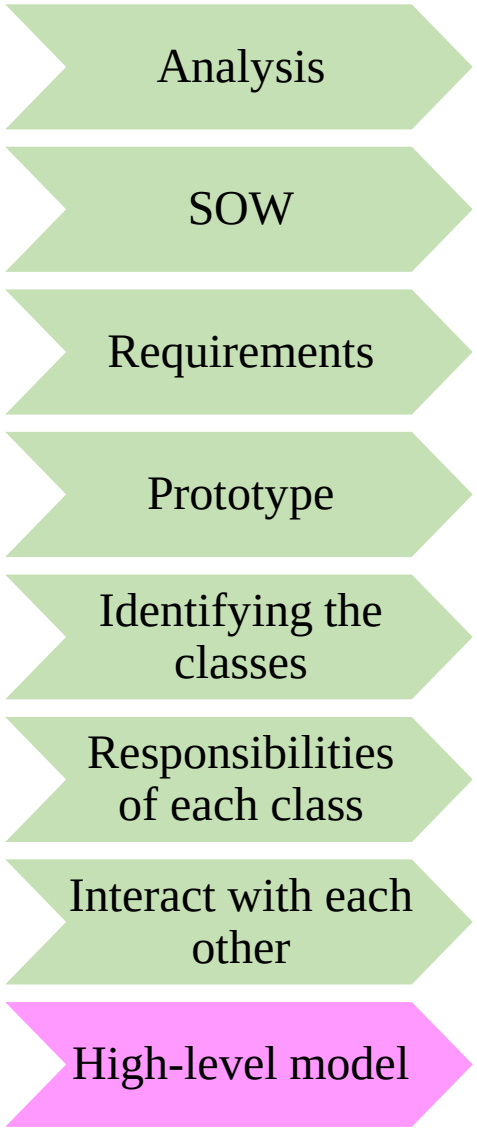
- Scope of work
  - A detailed description of the work, the software and hardware to be used, and the exact nature of the work.
- Location of the work
  - Where the location of the work to be done would be other than a standard location.
  - This would be applicable to an SOW for work to be performed off-shore.
- Period of performance
  - The start and finish date for the project, maximum billable hours per time period, etc.
- Deliverables schedule
  - Due dates for the deliverables of the project.
  - This would include completion dates for development, QA testing, User Acceptance Testing, etc.

Standard Checklist of SOW – 2

- Applicable standards
  - Industry standards or other standards imposed on the project deliverables.
  - These should include any standards such as ISO.
- Acceptance Criteria
  - These would include any quality standards that must be met.
  - They should also include any other conditions that must be met such as number of test cases, number of test cases executed, etc.
- Specialized Requirements
  - These will include any special qualifications for the work force.

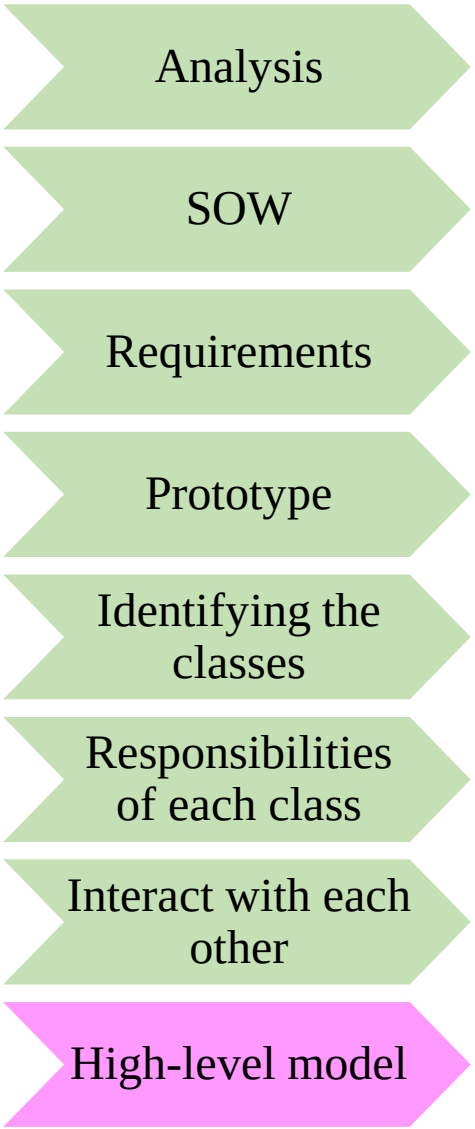
OO Design Process – Requirements

- Gathering the Requirements
  - The requirements are distilled from the SOW.
  - The requirements document describes what the users want the system to do.
  - The level of detail of the requirements document does not need to be of a highly technical nature,
  - The requirements must be specific enough to represent the true nature of the user's needs for the end product.
  - The requirements are usually represented as a summary statement or presented as bulleted items.
    - ※ The SOW is a document written in paragraph (even narrative) form.
  - The requirements are the final representation of the system that must be implemented.
    - ※ The SOW might contain irrelevant material.
  - All future documents in the software development process will be based on the requirements.



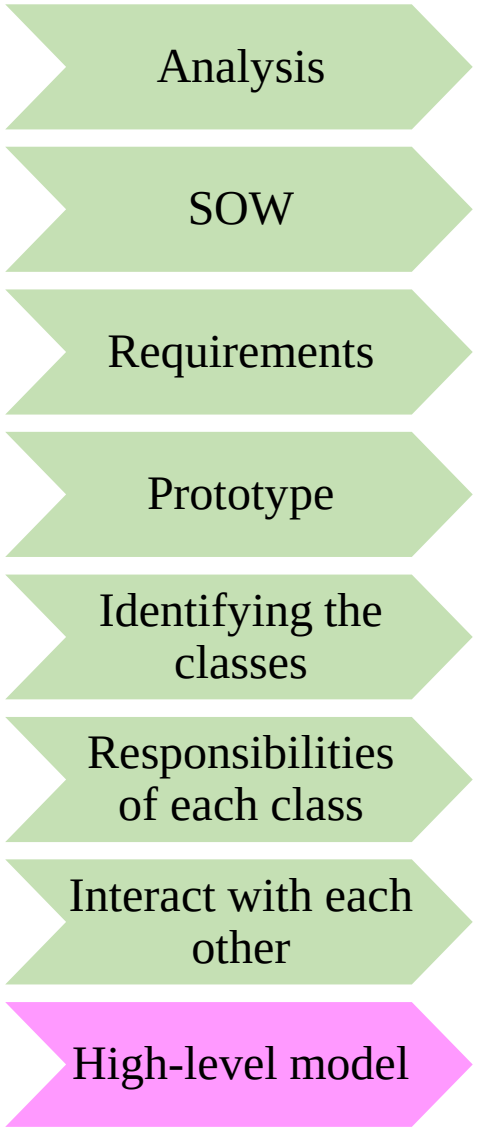
OO Design Process – Prototype

- Developing a Prototype of the User Interface
  - Creating a prototype is the one of the best ways to make sure users and developers understand the system.
  - A prototype can be just about anything; however, most people consider the prototype to be a simulated user interface.
  - By creating actual screens and screen flows, it is easier for people to get an idea of what they will be working with and what the system will feel like.
    - The look and feel of the user interface are the major concerns.
  - In any event, a prototype will almost certainly **not** contain all the functionality of the final system.
  - Most prototypes are created with an integrated development environment (IDE).
  - Having a good prototype can help immensely when identifying classes.



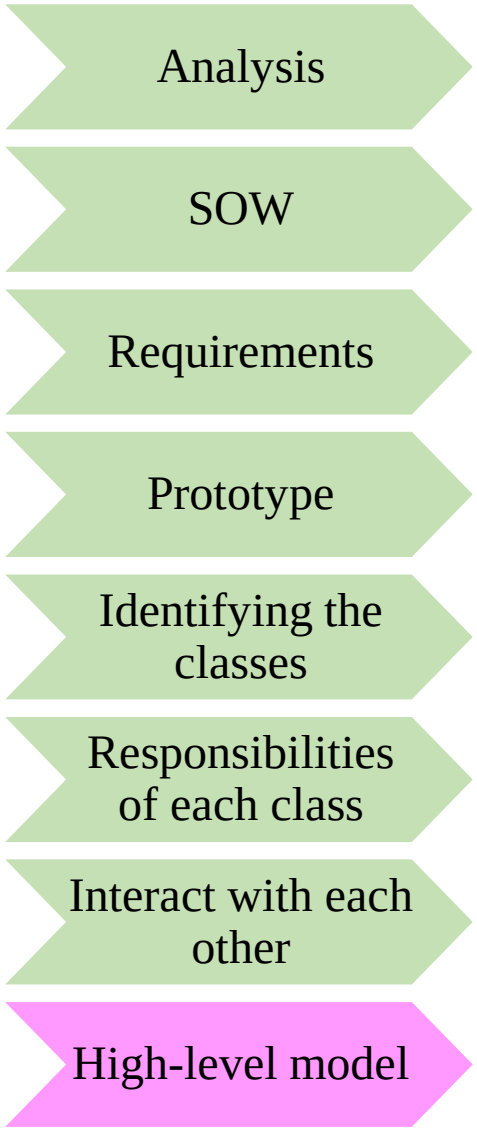
OO Design Process – Identifying the Classes

- Identifying the Classes
  - After the requirements are documented, the process of identifying classes can begin.
  - From the requirements, one straightforward way of identifying classes is to highlight all the nouns.
  - At various stages throughout the design, classes are eliminated, added, and changed.
    - Take advantage of the fact that the design is an iterative process throughout the design.



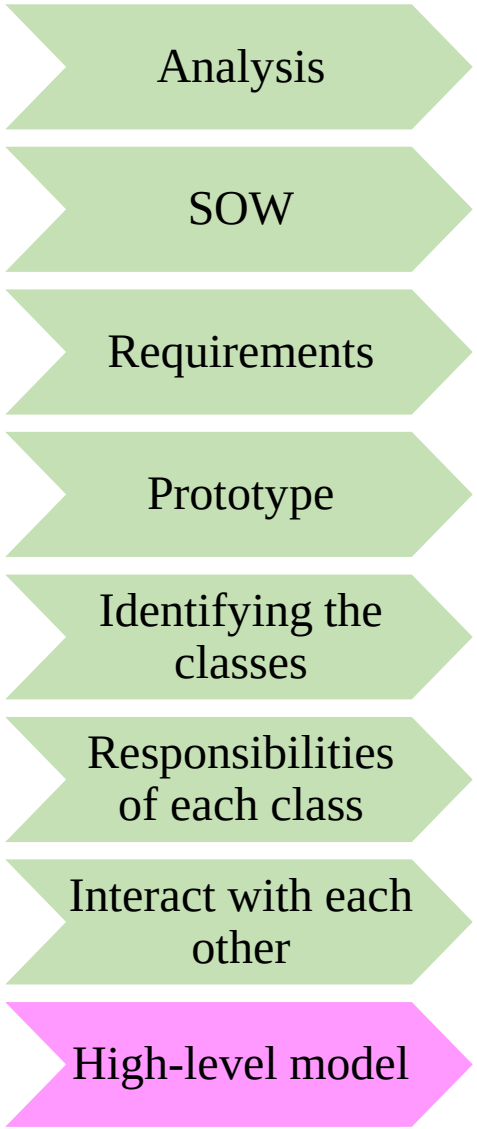
OO Design Process – Responsibilities of Each Class

- Determining the Responsibilities of Each Class
  - The responsibilities of each class
    - Data that the class must store
    - Operations the class must perform.
- Example
  - Responsibilities of an `Employee` object
    - Calculating payroll
    - Storing the various payroll rates
    - Transferring the money to the appropriate accounts
    - Storing the account numbers of various banks



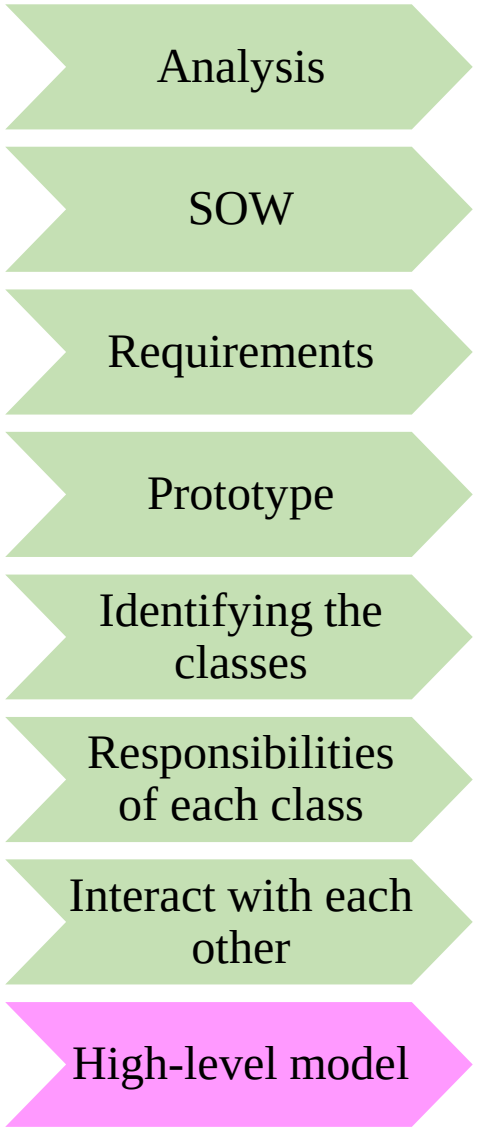
OO Design Process – Interact with Each Other

- Determining How the Classes Collaborate with Each Other
  - Most classes do not exist in isolation.
  - A class will have to interact with another class to get something it wants.
  - One class can send a message to another class when it needs information from that class, or if it wants the other class to do something for it.



OO Design Process – High-level Model

- Creating a Class Model to Describe the System
  - When all the classes are determined and the class responsibilities and collaborations are listed, a class model can be constructed.
- Class model
  - Represents the complete system.
  - Shows how the various classes interact within the system.
  - Example: UML(Unified Modeling Language)



Case Study
A Blackjack Example

Before Start

- Customer has come to you with a proposal
  - A very well-written SOW
  - A rulebook about how to play blackjack.
  - Goal : design a software system that will simulate the game of blackjack

Requirements Summary Statement (1/3)

- The intended purpose of this software application is to implement a game of blackjack.
  - In the game of blackjack, one or more individuals play against the dealer (or house).
  - Although there might be more than one player, each player plays only against the dealer, and not any of the other players.
- From a player's perspective, the goal of the game is to draw cards from the deck until the sum of the face value of all the cards equals 21 or as close to 21 as possible, without exceeding 21.
  - If the sum of the face value of all the cards exceeds 21, the player loses.
  - If the sum of the face value of the first two cards equals 21, the player is said to have blackjack.
  - The dealer plays the game along with the players.
  - The dealer must deal the cards, present a player with additional cards, show all or part of a hand, calculate the value of all or part of a hand, calculate the number of cards in a hand, determine the winner, and start a new hand.

Requirements Summary Statement (2/3)

- A card must know what its face value is and be able to report this value.
  - The suit of the card is of no importance (but it might be for another game in the future). All cards must be members of a deck of cards.
  - This deck must have the functionality to deal the next card, as well as report how many cards remain in the deck.
- During the game, a player can request that a card be dealt to his or her hand.
- The player must be able to display the hand, calculate the face value of the hand, and determine the number of cards in the hand.
  - When the dealer asks the player whether to deal another card or to start a new game, the player must respond.

Requirements Summary Statement (3/3)

- Each card has its own face value (suit does not factor into the face value). Aces count as 1 or 11. Face cards (Jack, Queen, King) each count as 10. The rest of the cards represent their face values.
- The rules of the game state that if the sum of the face value of the player's cards is closer to 21 than the sum of the face value of the dealer's cards, the player wins an amount equal to the bet that was made.
  - If the player wins with a blackjack, the player wins 3:2 times the bet made (assuming that the dealer does not also have blackjack).
  - If the sum of the face value of the player's cards exceeds 21, the bet is lost. Blackjack (an ace and a face card or a 10) beats other combinations of 21.
  - If the player and the dealer have identical scores and at least 17, it is considered a draw, and the player retains the bet

The next step

- Take the perspective/view of the user.
  - we are not interested in the implementation
  - we'll concentrate on the interface

- Study the requirements summary statement
- Start identifying the classes
- How we are going to model and track the classes that we ultimately identify

Using CRC Cards

- ***Class-Responsibility-Collaboration*** *cards (CRC)*
  - one of the most popular methods for identifying and categorizing classes
  - keep track of the classes as well as their interactions.
  - quite literally, a collection of standard index cards.
- Each CRC card
  - represents a single class's
    data attributes,
    responsibilities,
    and collaborations.
  - has three sections
    The name of the class
    The responsibilities of the class
    The collaborations of the class

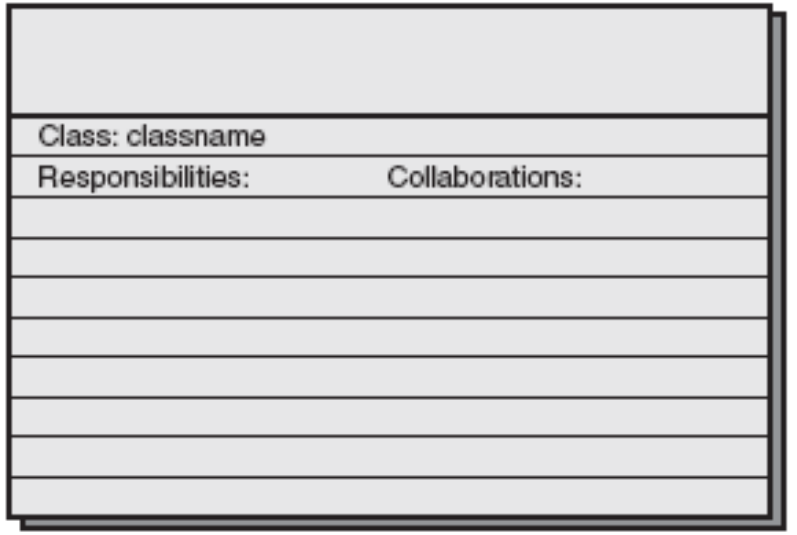


Figure 6.4 The format of a CRC card.

Identifying the Blackjack Classes (1/5)

- In general, classes correspond to nouns, which are objects - people, places, and things.
- Go through the requirements summary statement and highlight all the nouns
  - Good list from which you can start collecting objects.
  - Nouns are not the only places where classes are found.
- Not all the classes that you identify from the list of nouns or elsewhere will make it through to the final cut.
- Some classes that were not in your original list might actually make the final cut. – Note: "iterative process"

Identifying the Blackjack Classes (2/5)

- A list of the possible objects(classes) – starting point
  - Game
  - Blackjack
  - Dealer
  - House
  - Players
  - Player
  - Cards
  - Card
  - Deck
  - Hand
  - Face value
  - Suit
  - Winner
  - Ace
  - Face card
  - King
  - Queen
  - Jack
  - Game
  - Bet
- Begin the process of fine-tuning the list of classes
  - iterate through this a number of times and make changes

Identifying the Blackjack Classes (3/5)

- Explore each of the possible classes
  - **Game** : *Blackjack is the name of the game. Thus, we treat this in the same way we* treated the noun *game.*
  - ~~**Blackjack**~~ : *game might be considered a noun, but the game is actually* the system itself, so we will eliminate this as a potential class.
  - **Dealer** : Because we cannot do without a dealer, we will keep this one. There are enough additional attributes of a dealer.
  - ~~**House**~~ : another name for the dealer
  - ~~**Players**~~ and **player** : We want the class to represent a single player and not a group of players.
  - ~~**Cards**~~ and **card** : the same logic as *player.*
  - **Deck** : a lot of actions required by a deck (shuffling, drawing)

Identifying the Blackjack Classes (4/5)

- **Hand :** This game requires that a player has a single hand. Keep this class for extensibility.
  - theoretically possible for a player to have multiple hands
  - use the concept of a hand in other card games
- ~~**Face value**~~ : best represented as an attribute in the *card* class.
- ~~**Suit**~~ : *For the blackjack game, we do not need* to keep track of the suit. However, there are card games that need to keep track of the suit. Thus, to make this class reusable, we should track it. However, the suit is an attribute of a card.
- ~~**Ace**~~ : better be represented as an attribute of the card class
- ~~**Face Card**~~ : better be represented as attribute of the card class
- ~~King~~ : better be represented as attribute of the card class,
- ~~Queen~~ : better be represented as attribute of the card class,

Identifying the Blackjack Classes (5/5)

– Bet : presents a dilemma.

- The requirements statement includes a bet in the description
- A bet is not a logical attribute of a player.
- Abstracting out the bet is a good idea

because

we might want to bet

various things (money, chips, your watch, your horse…)

Figure 6.5 The initial blackjack classes.

Identifying the Classes' Responsibilities (1/12)

- Responsibilities relate to actions.
- You can generally identify responsibilities by selecting the verbs from the summary of the requirements
- Keep in mind the following:
  - Not all verbs in the requirements summary will ultimately end up as responsibilities.
  - Need to combine several verbs to find an actual responsibility.
  - Some responsibilities ultimately chosen will not be in the original requirements summary.
  - Because this is an iterative process, you need to keep revising and updating both the requirements summary and the responsibilities.
  - If two or more classes share a responsibility, each class will have the responsibility.

Identifying the Classes' Responsibilities (2/12)

- Initial list of the possible responsibilities for our classes:
  - Card
    - Know its face value
    - Know its suit
    - Know its value
    - Know whether it is a face card
    - Know whether it is an ace
    - Know whether it is a joker
  - Deck
    - Shuffle
    - Deal the next card
    - Know how many cards are left in the deck
    - Know whether there is a full deck to begin

Identifying the Classes' Responsibilities (3/12)

- Hand
  - Know how many cards are in the hand
  - Know the value of the hand
  - Show the hand
- Dealer
  - Deal the cards
  - Shuffle the deck
  - Give a card to a player
  - Show the dealer's hand
  - Calculate the value of the dealer's hand
  - Know the number of cards in the dealer's hand
  - Request a card (hit or hold)
  - Determine the winner
  - Start a new hand

Identifying the Classes' Responsibilities (4/12)

- Player
  - Request a card (hit or hold)
  - Show the player's hand
  - Calculate the value of the player's hand
  - Know how many cards are in the hand
  - Know whether the hand value is over 21
  - Know whether the hand value is equal to 21 (and if it is a blackjack)
  - Know whether the hand value is below 21
- Bet
  - Know the type of bet
  - Know the value of the current bet
  - Know how much the player has left to bet
  - Know whether the bet can be covered

Identifying the Classes' Responsibilities (5/12)

- Iterate through this a number of times and make changes
  - Card
    - Know its face value
      - The card definitely needs to know this.
      - From an interface perspective, call this *display face value*.
    - Know its suit
      - rename it *display name (which will identify the suit).*
      - *don't need this for blackjack*
      - keep it for potential reuse purposes.
    - ~~Know whether it is a face card.~~
      - The *report value* responsibility can probably handle this.
    - ~~Know whether it is an ace, Know whether it is a joker~~
      - Same as previous item

Identifying the Classes' Responsibilities (6/12)

- Deck
  - Shuffle
    - need to shuffle the deck.
  - Deal the next card
    - need to deal the next card
  - Know how many cards are left in the deck
    - At least the dealer needs to know if there are any cards left.
  - Know if there is a full deck to begin.
    - The deck must know whether it includes all the cards.
    - might be an internal implementation issue

Identifying the Classes' Responsibilities (7/12)

- Hand
  - Know how many cards are in the hand
    - need to know how many cards are in a hand.
    - From an interface perspective, rename this *report the number of cards in the hand.*
  - Know the value of the hand
    - need to know the value of the hand,
    - From an interface perspective, rename this *report the value of the hand.*
  - Show the hand
    - need to be able to see the contents of the hand.

Identifying the Classes' Responsibilities (8/12)

- **Dealer(1/2)**
  - Deal the cards
    - The dealer must be able to deal the initial hand.
  - Shuffle the deck
    - The dealer must be able to shuffle the deck.
  - Give a card to a player
    - The dealer must be able to add a card to a player's hand'
  - Show the dealer's hand
    - a general function for all players(including the dealer)
    - perhaps the hand should show itself and the dealer should request this.
  - Calculate the value of the dealer's hand
    - Same as previous.
    - "*calculate" is an implementation issue* → rename it "*show the value of the dealer's hand*".

Identifying the Classes' Responsibilities (9/12)

- **Dealer(2/2)**
  - Know the number of cards in the dealer's hand
    - same as *show the value of the dealer's hand*
    - *rename* it *show the number of cards in the dealers hand.*
  - Request a card (hit or hold)
    - A dealer must be able to request a card.
  - Determine the winner
    - depends on whether we want the dealer to calculate this or the game object. For now, let's keep it.
  - Start a new hand
    - keep this one for the same reason as the previous item.

Identifying the Classes' Responsibilities (10/12)

- **Player**
  - Request a card (hit or hold)
    - A player must be able to request a card
  - Show the player's hand
    - a general function for all players, so perhaps the hand should show itself and the dealer should request this.
  - Calculate the value of the player's hand
    - Same as previous.
    - *"calculate" is an implementation issue* → rename "*show the value of the player's hand*".
  - Know how many cards are in the hand
    - same as *show the player's hand*
    - rename it "*show the number of cards in the player's hand*".
  - Know whether the hand value is over 21, equal to 21, or below 21.
    - The player and dealer need to know this to make a decision about whether to request a card.

Identifying the Classes' Responsibilities (11/12)

- **Bet**
  - Know the type of bet
    - keep this for future reuse
    - For this game, the type of the bet is always money.
  - Know the value of the current bet
    - need this to keep track of the value of the current bet.
  - Know how much the player has left to bet
    - The bet can also act as the pool of money that the player has available.
  - Know whether the bet can be covered
    - a simple response that allows the dealer to determine whether the player can cover the bet.

Identifying the Classes' Responsibilities (12/12)

- After careful consideration, we decide that the bet class is not needed
- The decision needs to be based on two issues:
  - Do we really need the class now or for future classes?
  - Will it be easy to add later without a major redesign of the system?

UML Use-Cases: Identifying the Collaborations (1/6)

- Study the responsibilities and determine what other classes the object interacts with.
  - what other classes does this object need to fulfill all its required responsibilities and complete its job?
  - might find that you have missed some necessary classes or that some classes you initially identified are not needed → add
- To help discover collaborations, **use-case scenarios** can be used
  - A *use-case is a transaction* or sequence of related operations that the system performs in response to a user request or event.
  - The real purpose of creating use-case scenarios is to help you refine the choice of your classes and their responsibilities.
- For each use-case, identify the objects and the messages that it exchanges.
  - create **collaboration diagrams** to document this step

UML Use-Cases: Identifying the Collaborations (2/6)

- A single possible scenario with a dealer and a single player
  1. Dealer shuffles deck
  2. Player makes bet
  3. Dealer deals initial cards
  4. Player adds cards to player's hand
  5. Dealer adds cards to dealer's hand
  6. Hand returns value of player's hand to player
  7. Hand returns value of dealer's hand to dealer
  8. Dealer asks player whether player wants another card
  9. Dealer deals player another card
  10. Player adds the card to player's hand
  11. Hand returns value of player's hand to player
  12. Dealer asks player whether player wants another card
  13. Dealer gets the value of the player's hand
  14. Dealer sends or requests bet value from players
  15. Player adds to/subtracts from player's bet attribute

UML Use-Cases: Identifying the Collaborations (3/6)

Figure 6.6 Start the game.

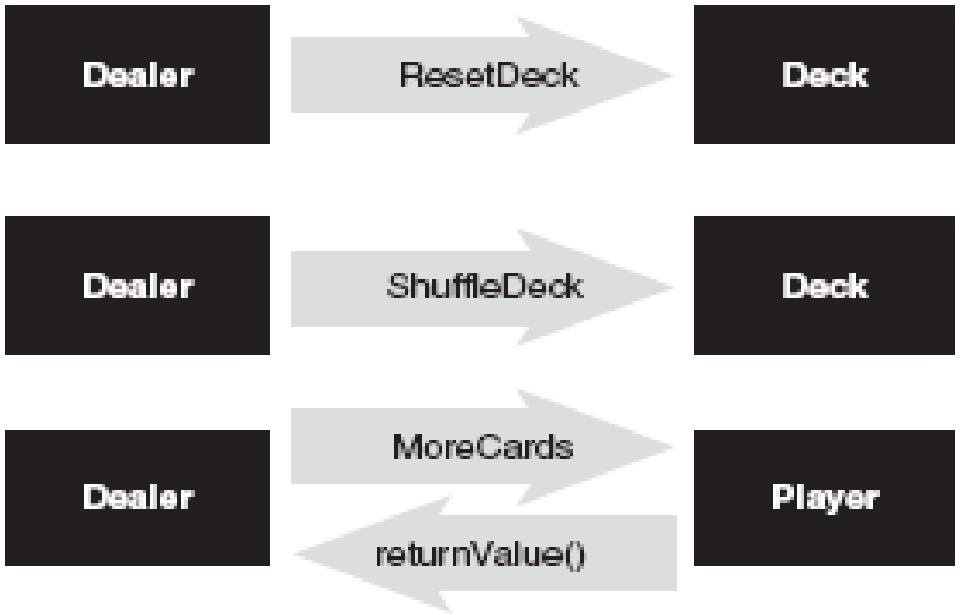


Figure 6.7 Shuffle and initially deal.

Figure 6.8 Get the hand value.

UML Use-Cases: Identifying the Collaborations (4/6)

Figure 6.9 Get a card.

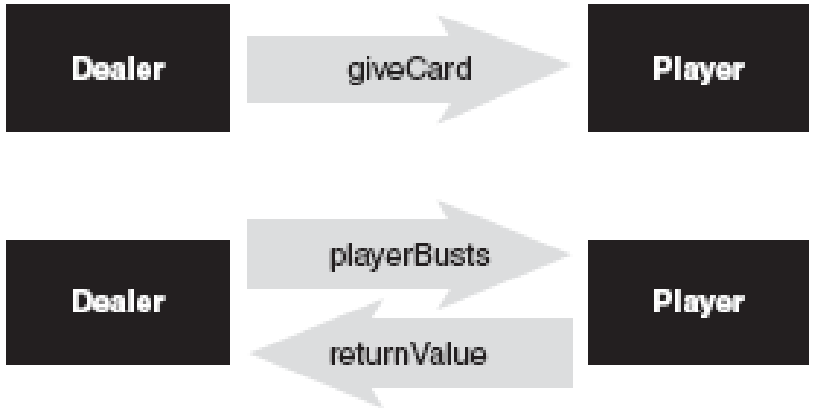


Figure 6.10 Deal a card and check to see whether the player busts.

Figure 6.11 Return the value of the hand.

UML Use-Cases: Identifying the Collaborations (5/6)

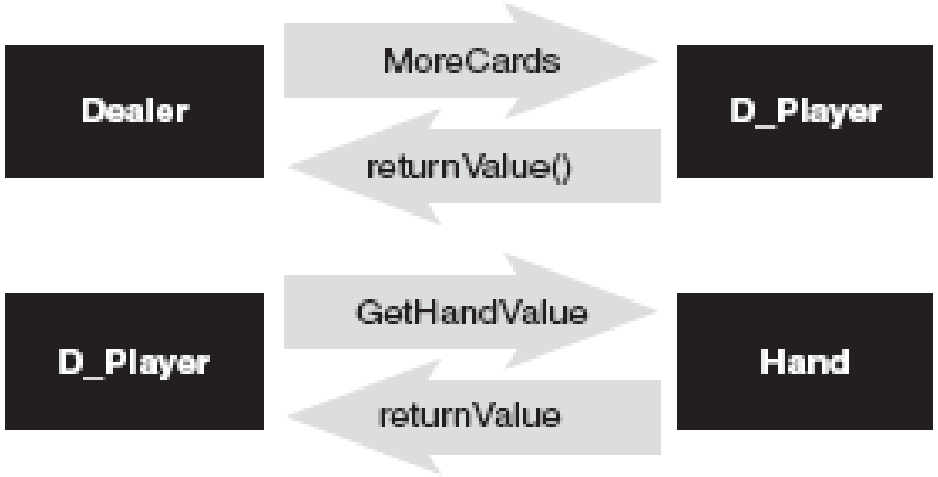


Figure 6.12 Does the dealer want more cards?

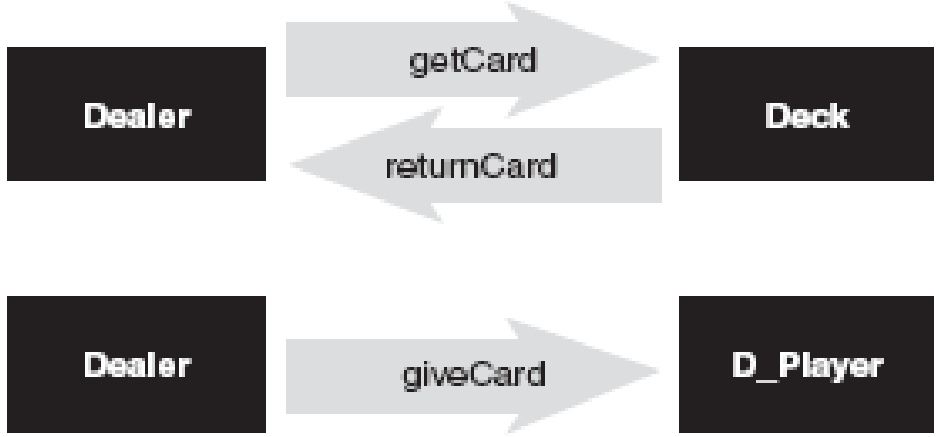


Figure 6.13 If requested, give the dealer a card.

UML Use-Cases: Identifying the Collaborations (6/6)

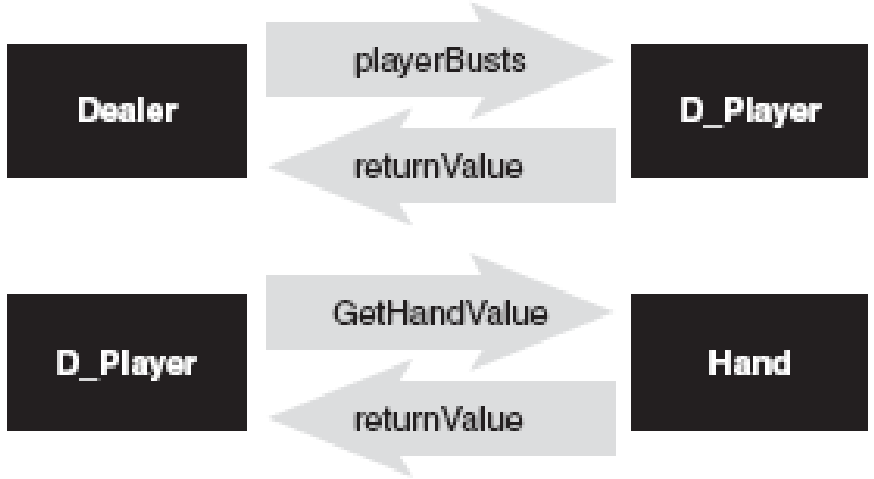


Figure 6.14 Does the dealer bust?

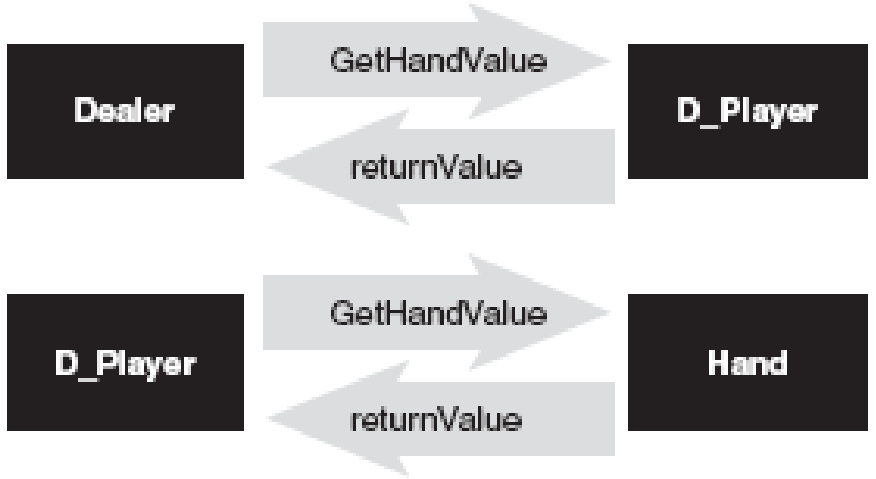


Figure 6.15 Do either the dealer or the player stand?

First Pass at CRC Cards (1/2)

- We have identified the initial classes and the initial collaborations → complete the CRC cards for each class
- Some CRC cards are appropriate to this initial design.

Class: Card	
Responsibilities:	Collaborations:
Get name	Deck
Get value	

Figure 6.16 A CRC card for the **`Card`** class.

Class: Deck	
Responsibilities:	Collaborations:
Reset deck	Dealer
Get deck size	Card
Get next card	
Shuffle Deck	
Show deck.	

Figure 6.17 A CRC card for the **`Deck`** class.

First Pass at CRC Cards (2/2)

Class: Dealer	
Responsibilities:	Collaborations:
Start a new game.	Hand
Get a card.	Player
	Deck

Figure 6.18 A CRC card for the **`Dealer`** class.

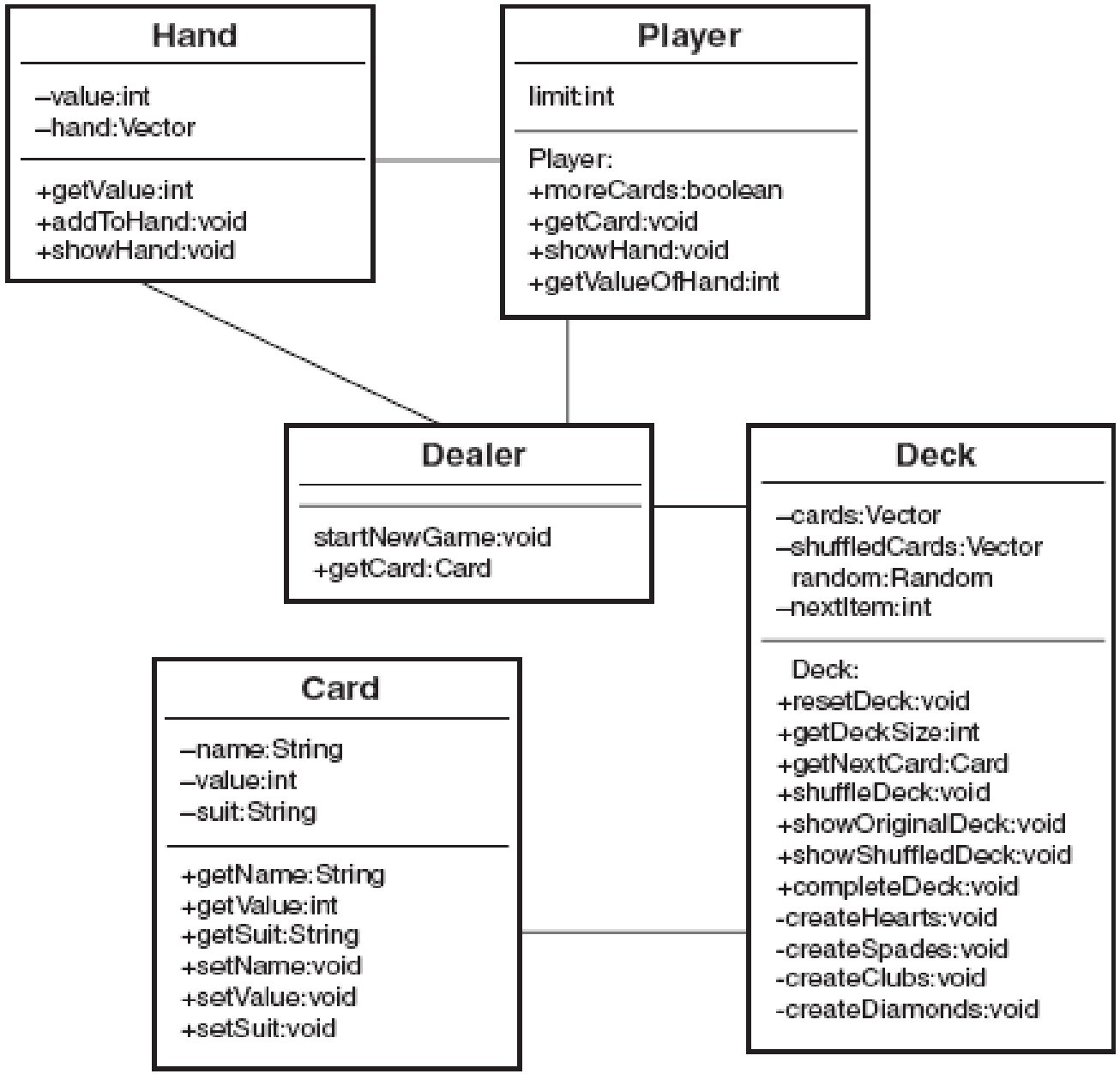
Class: Player	
Responsibilities:	Collaborations:
Want more cards?	Hand
Get a card.	Dealer
Show hand.	
Get value of hand.	

Figure 6.19 A CRC card for the **`Player`** class.

Class: Hand	
Responsibilities:	Collaborations:
Return Value	Player
Add a card	Dealer
Show Hand	

Figure 6.20 A CRC card for the **`Hand`** class.

UML Class Diagrams: The Object Model



The Next Step

- Prototyping the User Interface
  - final step in the OO design process
  - Create a prototype of our user interface.
  - Prototype will provide invaluable information to help navigate through the iterations of the design
  - There are several ways to create a user interface prototype.
  - You can sketch the user interface by simply drawing it on paper or a whiteboard.
  - You can use a special prototyping tool or even a language environment like Visual Basic, which is often used for rapid prototyping. Or you can use the IDE from your favorite development tool to create the prototype.
  - However you develop the user interface prototype, make sure that the users have the final say on the look and feel.

Summary

- OO design process:
  1. Doing the proper analysis
  2. Developing a statement of work (SOW) that describes the system
  3. Gathering the requirements from this statement of work
  4. Developing a prototype for the user interface
  5. Identifying the classes
  6. Determining the responsibilities of each class
  7. Determining how the various classes interact with each other
  8. Creating a high-level model that describes the system

Question?