Fundamentals of Object-Oriented Analysis and Design

Design – Fundamentals of OOAD

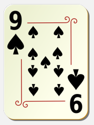
- OOAD – object-oriented analysis and development

The idea of object-oriented programming is that the organization of the program should match how you think about the problem.

- a program manipulates values that represent the ideas in the problem
  - classes reflect key concepts/things
    - often things which need some kind of representation (data storage) in the program, but classes can also exist to group together related functionality
  - instance variables store information about those things
  - methods provide ways to access/use/manipulate the stored information

Example

In a typical card game, each player gets a hand of cards. The deck is shuffled and cards are dealt one at a time from the deck and added to the players' hands. In some games, cards can be removed from a hand, and new cards can be added. The game is won or lost depending on the value (ace, 2, 3, ..., king) and suit (spades, diamonds, clubs, hearts) of the cards that a player receives.



- things
  - card
  - hand of cards
  - deck
- instance variables – relevant information about the things
  - card – value, suit
  - hand – the cards in the hand (and their order)
  - deck – the cards in the deck (and their order)
- methods – access/manipulation of the stored info
  - card – get value, get suit
  - hand – add card (at the end), add card at a position, get card at a position, remove card at a position
  - deck – shuffle, deal card

https://freesvg.org/nine-of-spades-playing-card-vector-illustration
https://commons.wikimedia.org/wiki/File:Hand_of_cards.jpg
https://www.publicdomainpictures.net/en/view-image.php?image=11873&picture=deck-of-cards

Example

In a typical card game, each player gets a hand of cards. The deck is shuffled and cards are dealt one at a time from the deck and added to the players' hands. In some games, cards can be removed from a hand, and new cards can be added. The game is won or lost depending on the value (ace, 2, 3, ..., king) and suit (spades, diamonds, clubs, hearts) of the cards that a player receives.

- things
  - card game
  - player
- instance variables – relevant information about the things
  - card game – the game elements (player's hands, deck, etc)
  - player – their hand
- methods – access/manipulation of the stored info
  - card game – the main program

→ “card game” is covered by the main program
→ “player” is sufficiently represented by just having their hands

Program Design

Other goals of object-oriented design –

- modularity
  - because small, independent chunks are easier to understand (and reuse)
- encapsulation and information hiding
  - because it is easier to understand a chunk if you don't have to deal with all the details of how it does what it does (*information hiding*)
  - because isolating design decisions means you can change your mind about them – or support multiple alternatives – without changing the rest of the program (*encapsulation*)

→ group related values together and protect the actual variables with methods to manipulate those values

Program Design Strategy

Goal: translate the concepts and language of the problem into classes and methods.

- identify classes – the kinds of things
  - a starting point is textual analysis – look for the nouns
    - but not every noun – some may be synonyms or things that don't need to be represented in the program
- identify instance variables – consider information storage
  - what values are needed to represent the class things in the program?
- identify the methods needed to access/manipulate the data that a class stores
  - a starting point is textual analysis – look for the verbs
  - consider what makes sense for the concept

Program Design Strategy, Continued

- complete the design
  - is all of the program's functionality accounted for?
    - e.g. main program
    - classes may also exist primarily to gather together related functionality
  - is everything the program needs to keep track of accounted for?
    - may be local variables in main or another class or additional instance variables in an already-identified class

Flip

Flip is a 2-player "strategic change-making game" published by Cheapass Games. You can find it, along with many other games, in the Game Preserve. (If you enjoy it, considering purchasing a copy.)

Each player starts with five dice. Roll the dice, and the player with the lowest total goes first. The players then alternate turns.

On each turn, the player either flips one of their own dice or plays one of their opponent's dice. However, players may *not* flip the same die twice without first playing one of the opponent's dice.

- *Flipping* means to turn the die to the opposite side of what is showing; since the top and bottom numbers of 6-sided dice always sum to 7, this means that the new value of a flipped die is 7 minus the old value. (6 flips to 1, 5 flips to 2, 4 flips to 3, 3 flips to 4, etc.)
- *Playing* means to put one of the opponent's dice into the middle of the table. They can then remove any number of dice whose total value does not exceed the value of the die played. For example, if a 6 is played, up to 5 points worth of dice can be removed from the middle.

The game continues until one player runs out of dice. The winner is the player with dice remaining, and the points scored is the sum of their dice.

Reset and continuing playing games until a player reaches 50 points.

Your program should display sufficient output so that the players can follow and play the game. In particular, the current game state (player 1's dice, player 2's dice, the dice in the middle, and which dice are flippable or not) should be displayed at the beginning of each turn, the winner and the current point totals for both players should be displayed at the end of each game, and the overall winner should be congratulated when 50 points is reached.

Your program should also perform appropriate error-checking and enforce the game rules. For example, players shouldn't be able to pick a non-existent die to flip or play, and players may not flip an unflippable die or remove too many points' worth of dice from the middle.

Flip

Design a program for the game Flip. Identify

- classes
- stored info (instance variables)
- methods

Flip Design

in progress

class	represents	stored info	methods
Player	one player in Flip		
Dice	one 6-sided die		
DiceCollection	collection of dice		create with 5 dice roll dice get total of dice flip a specific die

Class Design

- distinguish between a single thing and a collection of those things
  - naming convention is a singular name for the single thing and a plural name for the collection
  - a class for the single thing may not be needed if it is simple
    - type is represented by only one piece of information
    - an existing type matches well
  - a class for the collection may not be needed if an existing type serves well
    - e.g. an array for a collection of players that you iterate through to take turns
    - e.g. a Deck for a collection of cards since there are specialized actions with particular names like shuffling, dealing a card

Class Design

- a class should have a single simple well-defined purpose
  - don't omit writing this purpose down!
  - a long description, especially with lots of “and”, is a sign the class is probably too complex and/or not well-defined
- if you end up not identifying any stored information or any methods for a class, you probably don't need it
- different classes or just different objects?
  - is the type or nature of the stored information different, or is the difference really just in value?
  - are there different methods or different method bodies?

Class Design

- How do we know if the program design is complete, and on the right track?
  - do you have a specific clearly defined purpose for each class? do the instance variables and methods for a class fit that purpose?
    - need-to-know basis
  - for classes, have you accounted for all the things mentioned in text or when thinking about the problem?
    - when you have a collection of something, also consider the singular thing contained in the collection
  - for instance variables, have you accounted for everything that is important to represent about the kinds of objects you have classes for?
  - have you accounted for the main program functionality and the specific objects needed in the program?
  - getting the classes identified is more important than every last instance variable or method, especially getters

Class Design

- there are better designs and worse designs, but not always one perfect you're-wrong-with-anything-else design
  - weigh the considerations
    - e.g. modularity, information hiding, encapsulation – distinct logical units
    - e.g does the design reflect natural thinking about the problem?
      - naming, organization
      - kind of thing vs instance of thing