Motivation

Objects are the next step in organizing program and building modules –

- we can group subroutines and variables that together have a single whole purpose into an object, and treat that object like a bigger black box
  - in doing this, we define new types

The card and dice game programs from lab 8 and the war program from lab 9 provide examples of why this is useful.

- a class defines a kind of friend who can perform certain tasks
- an object is an actual friend standing there
  - you can have multiple objects / multiple friends capable of the same things
- you can outsource stuff to your friends
  - writing the program no longer means you being responsible for specifying every detail – you can focus on how to link together what the friends can do instead of also having to know how to do it

Object-Oriented Analysis and Development (OOAD)

- aka identifying classes
- it's better if each friend is responsible for a related collection of tasks
  - easier to figure out who to ask for what functionality – makes writing a program easier
  - more likely to be reusable – the same friend can be employed in future programs, too

Program Design With Classes

Advantages of object-oriented programming –

- more powerful black boxes (abstraction)

get suit and value

shuffle, deal, number of cards left

add and remove cards, get card (by position), sort by suit and value

- reusable software components

https://en.wikipedia.org/wiki/Playing_card
https://www.cuemath.com/data/card-probability/
https://www.freepik.com/free-vector/ace-diamonds-playing-card-isolated_21306216.htm

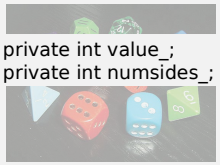
Program Design With Classes

- in the real world, there are various kinds of things…

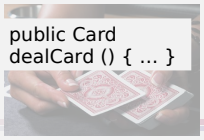
- …and you do stuff to manipulate those things

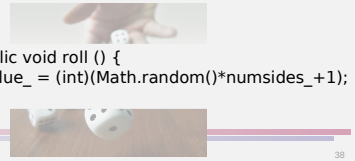
Program Design With Classes

- in the program, there are representations of things...



- …and stuff is done to manipulate those representations to mimic what the real-life action does to the real thing





Program Design With Classes

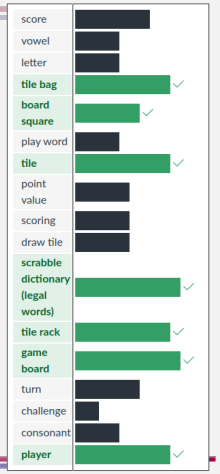
Developing a program –

- divide the work of the program up into modules
  - identify classes, then methods within those classes
  - main program – captures the flow of control that coordinates use of the modules
- develop pseudocode for the main program
- develop pseudocode for methods that need it as they are implemented
- translate the pseudocode into code

Class Design – Getting Started

- textual analysis – identify things that will be objects (nouns), with the kinds of things being candidates for classes
  - consider what is important about that thing in order to help weed out concepts that are more about process or that aren't necessary to represent

Class Design – Getting Started



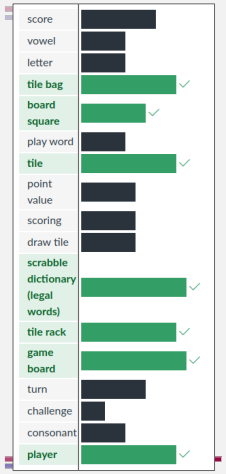
Class Design – Getting Started

- Scrabble
  - board square – tile in that position, scoring info
  - game board – the arrangement of board squares
  - tile rack – contents
  - tile bag – contents
  - tile – letter, point value
  - dictionary – the words that are legal to play
  - player – score, tiles

Class Design

- creating a new class is appropriate when…
  - the kind of thing is complicated – multiple pieces of information, complex and/or uncertain representation, complex tasks
  - there's not an existing type well-suited for the kind of thing
    - (full consideration of this point should wait until after the methods have been identified)

- Scrabble
  - board square – tile in that position, scoring info
    - multiple pieces of information (complex!) → BoardSquare class
  - game board – the arrangement of board squares
    - collection of stuff (complex!) → GameBoard class
  - tile rack – contents
    - collection of stuff (complex!) → TileRack class
  - tile bag – contents
    - collection of stuff (complex!) → TileBag class
  - tile – letter, point value
    - multiple pieces of information (complex!) → Tile class
  - dictionary – the words that are legal to play
    - collection of stuff (complex!) → Dictionary class
  - player – score, tiles
    - multiple pieces of information (complex!) → Player class



Class Design – Constructors and Methods

- a class has three kinds of elements
  - instance variables
  - constructors
  - methods
- "what's important about this thing?" focused on what needs to be represented → instance variables
- for constructors – you have a magic wand to conjure new objects into existence...what should they look like? (how should the instance variables be initialized?)
  - same initial value for all instances?
- for methods – textual analysis, focusing on verbs/actions that apply to the thing

- Scrabble
  - BoardSquare – tile contained, scoring info
    - create a particular kind (triple word score, etc)
    - place tile
  - GameBoard – tiles on the board and their arrangement
    - create empty board
    - play word (return score)
  - TileRack – contents
    - create empty rack
    - add tile
    - remove particular tile
  - TileBag – contents
    - create containing all tiles
    - draw tile (remove random tile)
  - Tile – letter, point value
    - create particular tile (with letter, point value)
  - Dictionary – the words that are legal to play
    - create containing all legal words
    - look up word (determine if a particular word is legal)
  - Player – score, tiles
    - create with no tiles, score 0
    - add to score

Completing the Design

- strive for a complete list of needed operations
  - review specifications, descriptions, etc to make sure no operations were missed (textual analysis)
  - write pseudocode to help identify program needs not present (or obvious) in the real-world version
    - e.g. printing the contents of the tile rack
  - think through the flow of information
    - e.g. the board manages the board squares, so we will need to ask the board to place a word rather than interacting directly with the squares
- complete the abstractions
  - e.g. getters to access stored information
  - e.g. ways to add to, remove from, and iterate through collections
  - some of these things may not be required by this particular program, but are helpful for future reusability

- Scrabble
  - BoardSquare – tile contained, scoring info
    - create a particular kind (triple word score, etc)
    - place tile
  - GameBoard – tiles on the board and their arrangement
    - create empty board
    - play word (return score)
    - **display board**
  - TileRack – contents
    - create empty rack
    - add tile
    - remove particular tile
    - **print tiles**
  - TileBag – contents
    - create containing all tiles
    - draw tile (remove random tile)
  - Tile – letter, point value
    - create particular tile (with letter, point value)
    - **get letter, get point value**
  - Dictionary – the words that are legal to play
    - create containing all legal words
    - look up word (determine if a particular word is legal)
  - Player – score, tiles
    - create with no tiles, score 0
    - manipulate set of tiles (add tile, remove particular tile)
    - add to score
    - **get score**