CS 5044
Object-Oriented Programming with Java

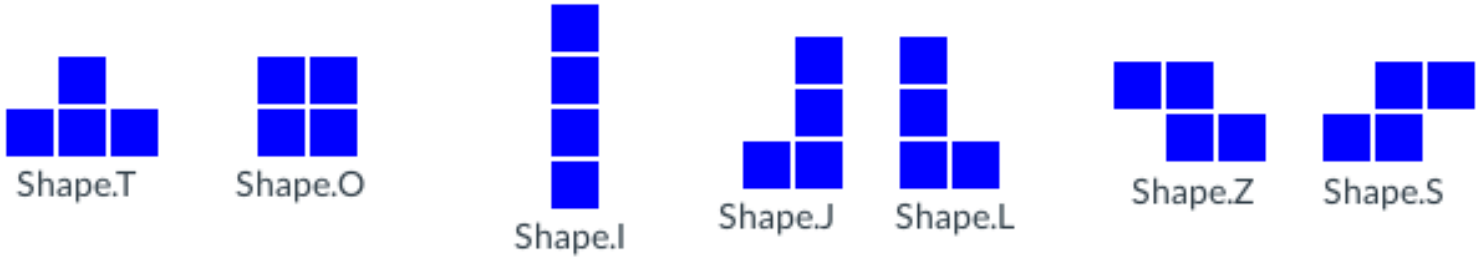
Q&A Session

Midterm information

- Midterm exam is split into three parts, all due Monday (10/14) at noon ET
  - Each part has an independent 30-minute time limit for 20 multiple-choice questions
  - Each part covers a cross-section of all materials from Module 1 through Module 6
  - Taken in Canvas, in the same format as our quizzes (but timed)
  - No external materials or references allowed

Meet the enums

- `enum Shape`
  - One value for each of the seven tetromino pieces:



  - Each Shape provides a Set of distinct Rotation values (default orientations shown above)
    - In Java, a `Set` is very similar to an `ArrayList`, but without any index values
      - There are a few other differences, but we'll explore those next week
      - For now, just use an *enhanced for()* loop to iterate over each element of the Set
  - You can also query the width of the shape (in blocks) after applying any valid Rotation
- `enum Rotation`
  - One value for each 90° rotation:
    - `NONE`
    - `CCW_90`
    - `CCW_180`
    - `CCW_270`

Game on

- `class Board`
  - Represents the state of the playing board at any given time
    - Class is immutable, meaning there are no public mutator methods
  - Provides public static constants: WIDTH and HEIGHT
    - Use these fields to avoid "magic numbers" (Programming Tip 4.1)
  - Contains a collection of fixed blocks we can query via getColumn(col)
    - Returns a boolean array, where true indicates the presence of a fixed block
  - We can ask the board to show us the hypothetical result of placing another piece
    - The piece will be placed and dropped, then any full rows will be cleared
    - Creates a new Board object; does not mutate the existing Board
  - We can also construct new Board objects with arbitrary blocks for testing
- `class Placement`
  - A Placement just holds together a Rotation value and a column index
  - This is what our AI needs to return to the game engine, for the given Shape:
    - First, the specified Rotation will be applied to the Shape
    - Next, the rotated Shape will be moved horizontally, such that...
      - ...the left-most block of the rotated Shape will be in the specified column
    - Invalid Placement objects are ignored by the game engine (with message to console)

Mind games

- `interface AI` (this is the interface we need to implement)
  - The primary method of interest is called by the game engine for each Shape
    - `public Placement findBestPlacement(Board currentBoard, Shape shape)`
  - The remaining methods must compute specific "cost" factors for a given board:
    - `public int getColumnHeightVariance(Board board)`
    - `public int getColumnHeightRange(Board board)`
    - `public int getAverageColumnHeight(Board board)`
    - `public int getTotalGapCount(Board board)`
  - The individual cost factor methods must be developed (and tested!) first
  - AFTER completing/testing the cost factor methods, start working on `findBestPacement()`
    - For every possible Placement (Rotation and column) of this Shape:
      - Get the board that would result from this hypothetical placement
      - Calculate cost factors for result board and combine with weights*
      - If this is the lowest overall cost so far, consider this placement as the new best
    - Return the best (minimum cost) Placement to the game engine
  - *Weights can all start at 1; this places **62.5** pieces, on average, over the 4 TEST modes
    - Then adjust the weights manually, using combinations of 0, 3, 6, and 9
      - The requirement is to place at least **125** pieces, on average, over the 4 TEST modes

Project 3: Hints and tips

- Other classes:
  - `ShapeStream` and `RandomMode` are needed for the OPTIONAL challenge only
  - `Tetris5044` is used only by the game engine itself; you won't ever need to use it
- Use meaningful variable names; it really makes a difference (and will be graded!)
  - For example, loop variables should be named `col` and `row` rather than `i` and `j`
- Beware of off-by-one errors in all loop bounds (use enhanced-for where possible)
  - The bottom-most row is 0, and the left-most column is 0 (beware `<` vs `<=` and similar)
- Account for *all* of the rows of the board when calculating costs
  - There are `Board.HEIGHT` rows to be considered
    - `Board.HEIGHT_LIMIT` is only useful for the OPTIONAL challenge
- Develop helper methods to reduce redundancy (and greatly simplify your code)
  - Consider a `getColumnHeight(Board board, int col)` helper:
    - Used by `getTotalGapCount()`, `getColumnHeightVariance()`, and `getColumnHeightRange()`
  - Consider a `getColumnBlockCount(Board board, int col)` helper:
    - Used to simplify `getTotalGapCount()`
- Let's share Eclipse again, time permitting...