Richard Guo
Dan Kim
Period 4
APCS

Our project consisted of several smaller projects that would cover various topics we learned over the semester. All except one (Election Campaign) were derived from problems on The Sedgewick and Wayne website. This file compiles all the header comments located in .java files for convenience, listing how to compile, execute each file and which files it depends on to run. UML Diagrams for Game of Life, Election Campaign, and Problem 1 from Percolation Case Study is at the bottom of this file.

Game of Life (Abstract classes with abstract methods, ArrayList, Loop)

```
/******************************************************************************
 *  Compilation:  javac Cell.java
 *  Execution:    N/A
 *  Dependencies: N/A
 *
 *  Abstract superclass for Alive and Dead Cells to be placed in the board.
 *
 ******************************************************************************/
```

```
/******************************************************************************
 *  Compilation:  javac Alive.java
 *  Execution:    N/A
 *  Dependencies: Cell.java
 *
 *  Subclass of Cell.java that provides implementation on how it
 *  should be shown on the board and change its state.
 *
 ******************************************************************************/
```

```
/******************************************************************************
 *  Compilation:  javac Dead.java
 *  Execution:    N/A
 *  Dependencies: Cell.java
 *
 *  Subclass of Cell.java that provides implementation on how it
 *  should be shown on the board and change its state.
 *
 ******************************************************************************/
```

```
/******************************************************************************
 *  Compilation:  javac Life.java
 *  Execution:    java Life N
 *  Dependencies: StdDraw.java Cell.java Alive.java Dead.java
 *
 *  Initializes a random board of Alive and Dead Cells.
 *  Every 0.5 seconds, updates a board of size N * N of Alive and Dead
 *  Cells according to Conway's rule on Game of Life.
 *
 *  % java Life 10
 *
 ******************************************************************************/
```

Election Campaign (Interface)

```
/******************************************************************************
 *  Compilation:  javac CampaignActions.java
 *  Execution:    N/A
 *  Dependencies: N/A
 *
 *  Interface of campaign actions for candidate to use to affect its stats
 *
 ******************************************************************************/

/******************************************************************************
 *  Compilation:  javac Candidate.java
 *  Execution:    N/A
 *  Dependencies: CampaignActions.java
 *
 *  Candidate Object with preset funding and poll percentage
 *
 ******************************************************************************/

/******************************************************************************
 *  Compilation:  javac CampaignTest.java
 *  Execution:    java CampaignTest
 *  Dependencies: CampaignActions.java Candidate.java
 *
 *  Prints out each candidate's statistics after performing the set
 *  of actions available
 *
 ******************************************************************************/
```

Problem 1 from Percolation Case Study (Classes with subclasses, 2D Array)

```
/*************************************************************************
 *  Compilation:  javac Problem1Matrix.java
 *  Execution:    N/A
 *  Dependencies: StdOut.java
 *
 *  Superclass for all parts of problem 1 that requires printing out two-
 *  dimensional boolean array, using * to represent true and a space to
 *  represent false, including row and column numbers.
 *
 *************************************************************************/

/*************************************************************************
 *  Compilation:  javac Problem1a.java
 *  Execution:    java Problem1a N
 *  Dependencies: Problem1Matrix.java StdOut.java
 *
 *  Takes N from the command line and draws an  N- by-N matrix with the entry
 *  in row i and column j set to true if i and j are relatively prime.
 *
 *  % java Problem1a 10
 *
 *      0123456789
 *  0    *
 *  1   **********
 *  2    * * * * *
 *  3    ** ** **
 *  4    * * * * *
 *  5    **** ****
 *  6    *   * *
 *  7    ****** **
 *  8    * * * * *
 *  9    ** ** **
 *
 *************************************************************************/

/*************************************************************************
 *  Compilation:  javac Problem1b.java
 *  Execution:    java Problem1b N
 *  Dependencies: Problem1Matrix.java StdOut.java
 *
 *  Takes N (where N is a power of 2 ) from the command line and
 *  draws a Hadamard matrix of order N.
 *
 *  % java Problem1b 8
 *
 *      01234567
 *  0   ********
 *  1   * * * *
 *  2   **  **
 *  3   *  **  *
 *  4   ****
 *  5   * *  * *
 *  6   **    **
 *  7   *  * **
 *
 *************************************************************************/

/*************************************************************************
 *  Compilation:  javac Problem1c.java
 *  Execution:    java Problem1c N
 *  Dependencies: Problem1Matrix.java StdOut.java
 *
 *  Takes N from the command line and draws an N- by-N matrix such with the
 *  entry in row N and column j set to true if the coefficient of x^k in
 *  (1+x)^N (binomial coefficiant) is odd.
 *
 *  % java Problem1c 10
 *
 *       0123456789
 *  0
 *  1    *
 *  2    *
 *  3    **
 *  4    * *
 *  5    ****
 *  6    *   *
 *  7    **  **
 *  8    * * * *
 *  9    ********
 *
 *************************************************************************/
```

Sierpinski (Recursive function, 1D Array)

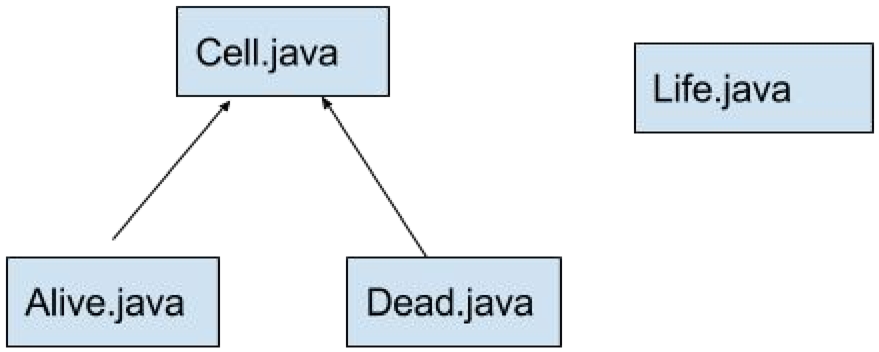
```
/*****************************************************************************
 *  Compilation:  javac Sierpinski.java
 *  Execution:    java Sierpinski N
 *  Dependencies: StdDraw.java
 *
 *  Plot an order N Sierpinski Triangle.
 *
 *  % java Sierpinski 5
 *
 *****************************************************************************
```

Stock Market (ArrayList)

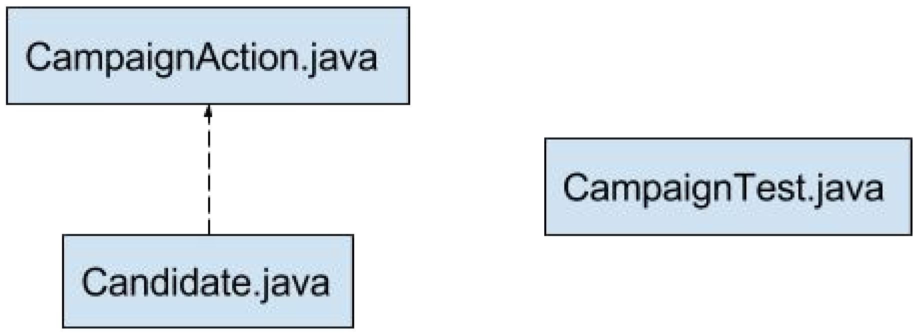
```
*****************************************************************************
 *  Compilation:  javac Invest.java
 *  Execution:    java Invest < stocks.txt
 *  Dependencies: StdOut.java StdIn.java
 *
 *  Creates a table that determines how much money you would have won or lost
 *  using Dilbert's rule given $10,000.00 cash and prices given in stocks.txt
 *
 *  % java Invest < stock15.txt
 *
 *  period   price     cash       shares    value
 *  ---------------------------------------------
 *  1    26.375   10000.00       0.00   10000.00
 *  2    25.500   10000.00       0.00   10000.00
 *  3    25.125   10000.00       0.00   10000.00
 *  4    25.000   10000.00       0.00   10000.00
 *  5    25.250       0.00     396.04   10000.00
 *  6    27.125       0.00     396.04   10742.57
 *  7    28.250       0.00     396.04   11188.12
 *  8    26.000   10297.03       0.00   10297.03
 *  9    25.500   10297.03       0.00   10297.03
 *  10    25.000   10297.03       0.00   10297.03
 *  11    25.125       0.00     409.83   10297.03
 *  12    25.250       0.00     409.83   10348.26
 *  13    26.375       0.00     409.83   10809.32
 *  14    25.500   10450.72       0.00   10450.72
 *  15    25.500   10450.72       0.00   10450.72
 *
 *****************************************************************************/
```

UML Diagrams

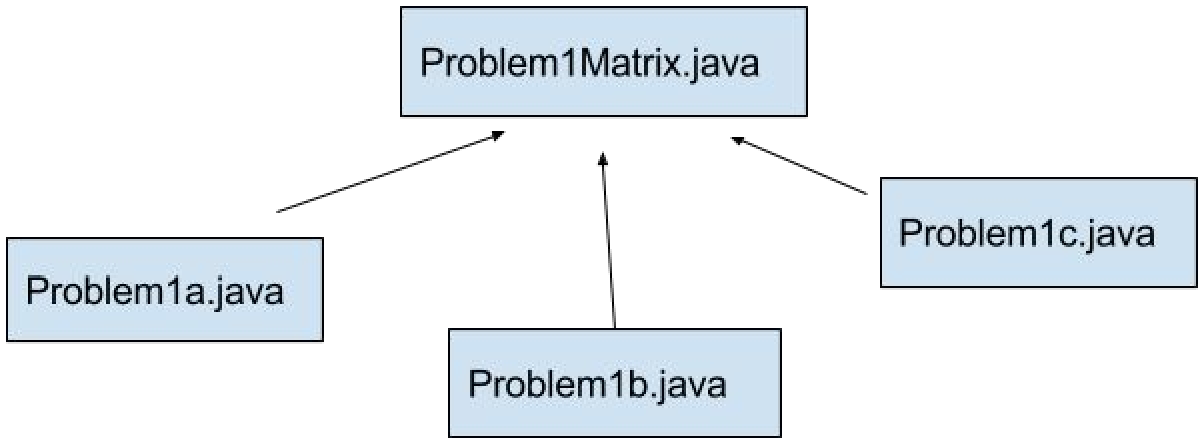
Game of Life



Election Campaign



Problem 1 from Percolation Case Study


Problem1Matrix.java
Problem1a.java
Problem1b.java
Problem1c.java