

## Problem Set 2

Due: 6pm Friday, February 23

### Revisions:

- In the URL used in the G00G stock quote example, the portion of the URL that was `table.csv` should be `table.csv`.
- As announced in lecture, the due date for this problem set has been extended until **11:59pm Mon. Feb. 26**.

### Notes:

- This is the final version of PS2 that includes Problems 2 and 3. Problem 1 has not changed since the first draft, but the `CS230Date` contract in Appendix A now has two new class methods: `min` and `max`.
- This assignment is due on **Friday, Feb. 23** rather than Tuesday, Feb. 20 (the date originally listed in the syllabus). After PS2, there will be only one other problem set (not two) before take-home Exam 1.

### Overview:

The purpose of this assignment is to give you practice with Input/Output (files, URLs, and keyboard), vectors, objects, and binary search. As in PS1, it is recommended that you (1) start early and (2) work with a partner (a different one than you worked with on PS1).

### Submission:

Each team should turn in a single hardcopy submission packet for all problems by slipping it under Lyn's office door by 1:30pm on the due date. The packet should include:

1. a team header sheet (see the end of this assignment for the header sheet) indicating the time that you (and your partner, if you are working with one) spent on the parts of the assignment.
2. your final version of `StockQuote.java` from Problem 1.
3. your final version of `StockInfo.java` from Problem 2.
4. your final version of `Portfolio.java` from Problem 3.
5. transcripts demonstrating that your `StockQuote`, `StockInfo` and `Portfolio` classes work as expected.

Each team should also submit a single softcopy (consisting of your entire final `ps2` directory) to the drop directory `~cs230/drop/ps2/username`, where `username` is the username of one of the team members (indicate which drop folder you used on your hardcopy header sheet). To do this, execute the following Linux commands in the team member's account where the code is stored:

```
cd /students/username/cs230
cp -R ps2 ~cs230/drop/ps2/username/
```

### Background:

All parts of this assignment deal with a stock portfolio management system based on real historical stock data that can be obtained from the Internet.

Based on the Java programming skills you displayed in Problem Set 1, You have been hired as an intern by Gopher Brokerage, Inc., to implement a simple stock portfolio management system for their customers. A stock portfolio specifies a collection of stocks along with the number of shares of each stock. The portfolio management system allows the customer to select a portfolio and find out information about the value of stocks in specified time periods.

In order to implement this system, you will need to access historical stock data from the Internet. Yahoo Finance is one of several web sites from which such data can be downloaded. The following URL is a template that can be used to download stock price information for a particular stock during a given range of time:

```
http://ichart.finance.yahoo.com/table.csv?s=<stock-symbol>&a=<start-month>
&b=<start-day>&c=<start-year>&d=<stop-month>&e=<stop-day>&f=<stop-year>
&g=<frequency-code>&ignore=.csv
```

In this template,

- `<stock-symbol>` is the stock exchange symbol for the stock, which is typically between one and four letters long. For example, the symbol for Google is GOOG, the symbol for Walmart is WMT, the symbol for Proctor & Gamble is PG, and the symbol for AT&T is T. You can look up stock symbols at <http://finance.yahoo.com/lookup>.
- `<start-month>` is the month of the desired starting date, which is denoted by an integer between 0 (January) and 11 (December).
- `<start-day>` is the day of the desired starting date, which is denoted by an integer between 1 and 31.
- `<start-year>` is the fully-specified 4-digit year of the desired starting date (e.g., 1986, 2007).
- `<stop-month>` is the month of the desired stopping date.
- `<stop-day>` is the day of the desired stopping date.
- `<stop-year>` is the year of the desired stopping date.
- `<frequency-code>` is a one-character code indicating whether daily (d), weekly (w), or monthly (m) information is desired.

For example, the URL used in the following example specifies daily stock information for Google from Dec. 22, 2006 to Jan. 16, 2007:

```
[fturbak@irwin utils] java FileOps urlToString "http://ichart.finance.yahoo.com/
table.csv?s=GOOG&a=11&b=22&c=2006&d=0&e=16&f=2007&g=d&ignore=.csv"
Date,Open,High,Low,Close,Volume,Adj Close
2007-01-16,507.55,513.00,503.30,504.28,7568900,504.28
2007-01-12,501.99,505.00,500.00,505.00,4473700,505.00
2007-01-11,497.20,501.75,496.18,499.72,7208200,499.72
2007-01-10,484.43,493.55,482.04,489.46,5968500,489.46
2007-01-09,485.45,488.25,481.20,485.50,5381400,485.50
2007-01-08,487.69,489.87,482.20,483.58,4754400,483.58
2007-01-05,482.50,487.50,478.11,487.19,6872100,487.19
2007-01-04,469.00,483.95,468.35,483.26,7887600,483.26
2007-01-03,466.00,476.66,461.11,467.59,7706500,467.59
2006-12-29,462.10,464.47,459.86,460.48,2559200,460.48
2006-12-28,467.12,468.58,462.25,462.56,3116200,462.56
2006-12-27,460.00,468.08,459.10,468.03,4231500,468.03
2006-12-26,456.52,459.47,454.59,457.53,2074300,457.53
2006-12-22,457.50,458.64,452.73,455.58,3988300,455.58
```

The information is provided in a tabular format known as *comma-separated-value (CSV)* format, in which each row is a line of text having columns fields that are separated by commas. The first line of the table lists titles for the columns and the remaining lines are the content rows of the table. In this assignment, you will focus on only the first and last columns. The first column specifies a date in the format YYYY-MM-DD<sup>1</sup>, while the last column gives the so-called *adjusted closing price*<sup>2</sup> of the stock for that date. For example, on Jan. 3, 2007, the adjusted closing price of Google stock was \$467.59.

Note that the information provided by Yahoo is in *reverse chronological* order — the more recent dates are listed earlier in the table. Also note that information is provided only for dates on

<sup>1</sup>In this format, months are 1-indexed even though they are 0-indexed in the URL. Go figure.

<sup>2</sup>The adjusted closing price may differ from the closing price because it takes into account details like dividends and stock splits.

which the stock exchange was open. In the above example, Dec. 22 is a Friday, and the exchange was closed for the Dec 23/24 weekend, Christmas (Mon. Dec. 25), the Dec 30/31 weekend, New Year's (Mon. Jan. 1 and Tue. Jan. 2), the Jan 6/7 weekend, the Jan 13/14 weekend, and Martin Luther King day (Mon. Jan. 15).

It appears that the database of stock prices supplied by Yahoo only goes back to 1962. Yahoo ignores any dates specified before this time and any dates specified after the current one. So you can get all daily stock information for a given stock by specifying any year before 1962 as the `<start-year>` (e.g., 1900), any year after the current year as the `<stop-year>` (e.g., 2008), and `d` as the `<frequency-code>`. For example:

```
[fturbak@irwin utils] java FileOps urlToString "http://ichart.finance.yahoo.com/
table.csv?s=G00G&a=0&b=1&c=1900&d=0&e=1&f=2008&g=d&ignore=.csv"
Date,Open,High,Low,Close,Volume,Adj Close
2007-02-13,459.15,462.78,457.26,459.10,4062600,459.10
2007-02-12,460.68,462.39,455.02,458.29,5754500,458.29
... many lines omitted ...
2004-08-20,101.01,109.08,100.50,108.31,11428600,108.31
2004-08-19,100.00,104.06,95.96,100.34,22351900,100.34
```

Google stock was introduced on Aug. 19, 2004, so the records don't go back any farther than this.

The problems of this assignment lead you through the construction of a simple stock portfolio manager that uses data from the Yahoo finance site. You should begin the problem by performing a CVS update:

```
cd ~/cs230
cvs update -d
```

This will install the `~/cs230/ps2` directory in your home directory. The `ps2` directory contains several files for this assignment. You will also be creating some new files from scratch in this directory.

### Problem 1 [20]: The StockQuote Class

A *stock quote* is the price of a stock at a specified point in time. In this assignment, a stock quote will always refer to the adjusted closing price of a stock on a particular day.

In this problem, your task is to create a `StockQuote` class in the file `~/cs230/ps2/StockQuote.java` that implements the contract specified in Fig. 1. Each instance of the `StockQuote` class simply pairs a date with a stock price on that date. (The name of the stock is not mentioned in the `StockQuote` instance.) Dates are represented as instances of the `CS230Date` class, which has already been implemented for you. See Appendix A for the `CS230Date` class contract.

Below are some sample statements and expressions involving `StockQuote` instances. The notation  $E \Rightarrow V$  means that expression  $E$  evaluates to value  $V$ .

```
StockQuote q = new StockQuote(CS230Date.fromString("2007-02-14", 230.06));
q.date().toString() => "2007-02-14"
q.price() => 230.06
q.toString() => "2007-02-14:230.06"
StockQuote q2 = new StockQuote(CS230Date.fromString("2007-02-13", 111.00));
StockQuote q3 = new StockQuote(CS230Date.fromString("2007-02-14", 110.00));
StockQuote q4 = new StockQuote(CS230Date.fromString("2007-02-14", 230.06));
StockQuote q5 = new StockQuote(CS230Date.fromString("2007-02-14", 251.00));
StockQuote q6 = new StockQuote(CS230Date.fromString("2007-02-15", 110.00));
q.equals(q2) => false ; Similar if q2 is replaced by q3, q5, or q6
q.equals(q4) => true
q.compareTo(q2) => a positive integer ; similar if q2 is replaced by q3
q.compareTo(q4) => 0
q.compareTo(q5) => a negative integer ; similar if q5 is replaced by q6
; From low to high, the above quotes are ordered: q2, q3, q=q4, q5, q6
```

In addition to implementing all methods in Fig. 1, you should write a `main()` method that tests

An instance of the `StockQuote` class represents a pair of a date (an instance of `CS230Date`) and a stock price (a `double`).

*Public Constructor Method:*

```
public StockQuote (CS230Date date, double price);
```

Creates a quote with date `date` and price `price`.

*Public Instance Methods:*

```
public double price ();
```

Returns the price of this quote.

```
public CS230Date date ();
```

Returns the date of this quote.

```
public String toString ();
```

Returns a string representation of this quote having the form `dateString:priceString`, where `dateString` is a string representation of this quote's date and `priceString` is a representation of this quote's price.

```
public boolean equals (Object x);
```

Returns `true` if `x` is a `StockQuote` with the same date and price as this quote, and `false` otherwise.

```
public int compareTo (StockQuote x);
```

Returns a negative number if this quote comes before `x` in the quote ordering (see the definition of quote ordering below), 0 if this quote is equal to `x` in the quote ordering, and a positive number if this quote comes after `x` in the quote ordering.

The ordering on quotes is a **lexicographic ordering** (i.e., dictionary ordering) in which quotes are first compared by date, and then by price. That is, quotes are first compared according to their dates; a quote with an earlier date precedes one with a later date, regardless of their prices. If the two dates are the same, then the quote with the lower price precedes the quote with the higher price.

*Public Class Methods:*

```
public static StockQuote fromLine (String line);
```

Returns a stock quote from a string that is one row of a CSV-formatted stock information table supplied by Yahoo. Here is an example of such a string:

```
"2007-01-16,507.55,513.00,503.30,504.28,7568900,504.28"
```

For this string, `fromLine` should return an instance of `StockQuote` whose date is 2007-01-16 and whose price is 504.28.

Figure 1: The contract for the `StockQuote` class.

your methods. For this part, your hardcopy submission should consist of (1) your final version of `StockQuote.java` and (2) a transcript of tests carried out using your `main()` method.

## Problem 2 [40]: The StockInfo Class

In this problem, your task is to create a `StockInfo` class in the file `~/cs230/ps2/StockInfo.java` that implements the contract specified in Figs. 2 and 3. Conceptually, each instance of the `StockInfo` class has four pieces of information:

1. The name of the stock (e.g. “General Motors” or “Google”);
2. The stock-ticker symbol of the stock (e.g. `GM` or `GOOG`).
3. A number of shares of that stock.
4. The historical daily adjusted-closing-price quotes for the stock, as determined from the Yahoo finance site (as explained earlier in the Background section). For example, this site has 628 quotes for Google from 2004-08-19 to 2007-02-15.

*Notes:*

- Include the following import statements at the top of `StockInfo.java`:

```
import java.io.*; // Allows use of BufferedReader, InputStreamReader
import java.util.*; // Allows use of Vector
import java.net.*; // Allows use of URL
```

- You should use a `Vector` of `StockQuote` objects to store the quote information in a `StockInfo` instance. Because Yahoo supplies these in reverse chronological order, it is convenient to store them sorted reverse chronological order as well. Don’t forget that the first line of a Yahoo stock table is a list of titles of the form `Date,Open,High,Low,Close,Volume,Adj Close`.
- You should read the historical stock data from the Yahoo finance site using the website-reading tools described in the lecture #4 notes. You should *not* use `FileOps.urlToString` as a black box, but should use the “parts” of `FileOps.urlToString` to read one line at a time from a web site.
- Your `getQuote()` method should use binary search to find the appropriate quote for the given date. It is helpful to express the binary search for `date` as an iteration in two state variables: `lo` (the low index) and `hi` (the high index) that satisfies the following invariants:
  - All quotes at indices less than `lo` have dates greater than `date`.
  - All quotes at indices greater than `hi` have dates less than `date`.

(Because the quotes are stored in reverse chronological order, dates get smaller as the index gets larger.)

Since implementing binary search can be challenging, you may want to first implement `getQuote()` using linear search and then implement binary search when everything else is working.

- Use `FileOps.keyboardToLine` to read a line of input from the console.
- Feel free to define any private helper methods that you find useful.
- A test version of `StockInfo.class` is provided in the `test` subdirectory of the `ps2` directory. You can use it to test the behavior of the `StockInfo` program.
- For this part, your hardcopy submission should consist of (1) your final version of `StockInfo.java` and (2) a transcript of tests carried out using your `main` method.

An instance of the `StockInfo` class represents a certain number of shares of a stock along with the historical daily adjusted-closing-price quotes for the stock.

*Public Constructor Method:*

```
public StockInfo (String name, String symbol, int shares);
```

Creates a `StockInfo` instance with the specified name, stock-ticker symbol, and number of shares. The instance returned by this constructor also has the historical daily adjusted-closing-price quotes for the stock as determined from the Yahoo finance site. Throws a `RuntimeException` if no stock with the symbol `symbol` is in the Yahoo database.

*Public Instance Methods:*

```
public String name ();
```

Returns the name of this stock.

```
public String symbol ();
```

Returns the stock-ticker symbol for this stock.

```
public int shares ();
```

Returns the number of shares for this stock.

```
public CS230Date firstDate ();
```

Returns the earliest date for which quote information is known about this stock. If no quote information is known, returns `CS230Date.MAX_DATE`.

```
public CS230Date lastDate ();
```

Returns the latest date for which quote information is known about this stock. If no quote information is known, returns `CS230Date.MIN_DATE`.

```
public int numQuotes ();
```

Returns the number of quotes known for this stock.

```
public StockQuote getQuote (CS230Date date);
```

If the `date` is not in the range of known quotes for this stock, returns `null`. If `date` is in the range known for the stock, returns the quote for that date. If there is no quote for the exact date, returns the quote for the nearest date preceding `date`. E.g., if the quotes cover all of 2006, then the query 2006-12-25 should return a quote for 2006-12-22, since there are no quotes for 2006-12-23 (a Saturday), 2006-12-23 (a Sunday), or 2006-12-25 (Christmas).

```
public String toString ();
```

Returns a string representation of this stock having the form

```
[StockInfo: shares shares of name (sym), num quotes between first and last]
```

where *shares* is the number of shares, *name* is the name, *sym* is the symbol, *num* is the number of known quotes, *first* is the earliest date of the known quotes, and *last* is the latest date of the known quotes

Figure 2: The contract for the `StockInfo` class, part 1.

*Public Class Methods:*

```
public static void main (String[] args);
```

There should be three elements in `args`: (1) a stock name, (2) a stock symbol, and (3) a number of shares – e.g., `java StockInfo "General Motors" GM 125`. When invoked in this manner, the `main()` method should (1) create a `StockInfo` instance `info` based on the arguments; (2) display `info.toString()`; and (3) enter a **read-eval-print loop (REPL)** that allows the user to interactively test `info`.

In a REPL, the user is interactively prompted for input and the program responds with the appropriate output. (The Linux shell is an example of a REPL.) In this case, the prompt should be the stock symbol of `info` followed by `>` and a space. E.g., the prompt for General Motors is `"GM> "`. This REPL understands three kinds of inputs:

1. If the input is a single date `date` (written in the standard `CS230Date` format) that is in the range of quotes known by `info`, the output is the `StockQuote` information for that date – i.e., the result of invoking `info.getQuote(date)`. If `date` is outside the known range of dates, an error message is displayed.
2. If the input is a pair of space-separated dates `start` and `stop`, the output is the `StockQuote` information (one quote per line) for all dates between `start` and `stop` for which such information exists.
3. If the input is `quit`, the REPL is terminated.

If the input is not one of the above forms, or if the dates specified are not in the right format, then a help message indicating the acceptable forms of input is displayed.

See Fig. 4 for a transcript of a sample REPL for `StockInfo`. In the example, help messages are displayed for `???` (not a valid input), `02/15/2007` (invalid date format), `2007-1-16` (invalid date format), and `2007-02-12 2007-02-13 2007-02-14` (three dates are not a valid input). Note that the quote given for `2006-12-25` (last Christmas) is dated `2006-12-22`, since this is the most recent quote for that date. The quote listing for the range `2004-01-01 2004-08-31` excludes all quotes before `2004-08-19` (they don't exist) as well as quotes for `2004-08-21,22,28,29` (weekend days).

Figure 3: The contract for the `StockInfo` class, part 2.

```

[fturbak@irwin ps2] java StockInfo Google GOOG 100
[100 shares of Google (GOOG), 628 quotes between 2004-08-19 and 2007-02-15]

GOOG> ???
Type one date (YYYY-MM-DD) in the range [2004-08-19, 2007-02-15]
  to get the nearest quote for that date.
Type two dates to get a listing of all quotes between those dates.
Type quit to exit this read-eval-print loop.

GOOG> 2004-08-19
2004-08-19:100.34

GOOG> 2007-02-15
2007-02-15:461.47

GOOG> 2006-12-25
2006-12-22:455.58

GOOG> 2007-08-18
The date 2007-08-18 is outside the range [2004-08-19, 2007-02-15] of known quotes.

GOOG> 2007-02-16
The date 2007-02-16 is outside the range [2004-08-19, 2007-02-15] of known quotes.

GOOG> 02/15/2007
Type one date (YYYY-MM-DD) in the range [2004-08-19, 2007-02-15]
  to get the nearest quote for that date.
Type two dates to get a listing of all quotes between those dates.
Type quit to exit this read-eval-print loop.

GOOG> 2007-1-16
Type one date (YYYY-MM-DD) in the range [2004-08-19, 2007-02-15]
  to get the nearest quote for that date.
Type two dates to get a listing of all quotes between those dates.
Type quit to exit this read-eval-print loop.

GOOG> 2004-01-01 2004-08-31
2004-08-19:100.34
2004-08-20:108.31
2004-08-23:109.4
2004-08-24:104.87
2004-08-25:106.0
2004-08-26:107.91
2004-08-27:106.15
2004-08-30:102.01
2004-08-31:102.37

GOOG> 2007-02-12 2010-01-01
2007-02-12:458.29
2007-02-13:459.1
2007-02-14:465.93
2007-02-15:461.47

GOOG> 2010-01-01 2007-02-02

GOOG> 2007-02-12 2007-02-13 2007-02-14
Type one date (YYYY-MM-DD) in the range [2004-08-19, 2007-02-15]
  to get the nearest quote for that date.
Type two dates to get a listing of all quotes between those dates.
Type quit to exit this read-eval-print loop.

GOOG> quit

[fturbak@irwin ps2]

```

### Problem 3 [40]: The Portfolio Class

In this problem, your task is to create a `Portfolio` class in the file `~/cs230/ps2/Portfolio.java` that implements the contract specified in Figs. 5 and 6. Conceptually, each instance of the `Portfolio` class has four pieces of information:

1. The name of the portfolio (e.g. “Technology” or “BlueChip Stocks”);
2. A collection of stocks (represented by `StockInfo` instances).
3. The first date for which all of the stocks in the collection have a quote.
4. The last date for which all of the stocks in the collection have a quote.

*Notes:*

- The order of stocks in a portfolio should be the same order in which they were added by `addStock()`.
- The `getValue()` method returns a non-negative value for any date between `firstDate()` and `lastDate()`, inclusive. In particular, it returns a non-negative value for days on which there is no quote for individual stocks (such as weekend days and holidays). In this case, it uses the most recent closing price.
- In your `fromString()` and `verboseValueString()` methods, you should use `StringBuffer` to accumulate a string value. As explained in lecture, this is more efficient than accumulating a string via string concatenation.
- Feel free to define any private helper methods that you find useful.
- A test version of `Portfolio.class` is provided in the `test` subdirectory of the `ps2` directory. You can use it to test the behavior of the `Portfolio` program.
- As indicated in the transcript in Appendix B, the files `small.txt`, `technology.txt`, and `bluechip.txt` contain sample stock portfolios. These are included in your `ps2` directory.
- For this part, your hardcopy submission should consist of (1) your final version of `Portfolio.java` and (2) a transcript of tests carried out using your `main` method.

An instance of the `Portfolio` class represents a named collection of `StockInfo` instances .

*Public Constructor Method:*

```
public Portfolio (String name);
```

Creates a `Portfolio` instance with the specified name whose collection of stocks is empty.

*Public Instance Methods:*

```
public String name ();
```

Returns the name of this portfolio.

```
public void addStock (StockInfo stock);
```

Adds `stock` to the stocks in this portfolio.

```
public CS230Date firstDate ();
```

Returns the earliest date for which a quote exists for all stocks in this portfolio.

```
public CS230Date lastDate ();
```

Returns the last date for which a quote exists for all stocks in this portfolio.

```
public double getValue (CS230Date date);
```

Returns the value of this portfolio on a particular date. The value of the portfolio is the sum of the share-weighted prices of all the the stocks in the portfolio on the given date. If the date is not in the range covered by the portfolio, returns -1.0. If the date is in the range covered by the portfolio, returns the value of the portfolio on that date.

```
public String toString ();
```

Returns a string representation of this portfolio having the form

```
[Portfolio: portfolio-name (date range [first, last):
    shares1 shares of stock-name1 (sym1)
        :
    sharesn shares of stock-namen (symn)
]
```

where *portfolio-name* is the name of the portfolio, *first* is the earliest date of the known quotes for all stocks in the portfolio, *last* is the latest date of the known quotes for all stocks in the portfolio, *shares*<sub>*i*</sub> is the number of shares for the *i*th stock in the portfolio, *stock-name*<sub>*i*</sub> is the name of the *i*th stock in the portfolio, and *sym*<sub>*i*</sub> is the stock-ticker symbol for the *i*th stock in the portfolio,

```
public String verboseValueString (CS230Date date);
```

Returns a string showing the value of this portfolio and its individual stocks on the given date. This string has the following form:

```
Portfolio value on date = value:
    shares1 shares of sym1 @ share-price1 = stock-price1)
        :
    sharesn shares of symn @ share-pricen = stock-pricen)
```

where *date* is the date specified by `date`, *value* is the value of the portfolio on the date `date`, *shares*<sub>*i*</sub> is the number of shares for the *i*th stock in the portfolio, *sym*<sub>*i*</sub> is the stock-ticker symbol for the *i*th stock in the portfolio, *share-price*<sub>*i*</sub> is the per-share price of the *i*th stock in the portfolio on the date `date`, and *cdstock-price*<sub>*i*</sub> is the total price (per-share price times number of shares) of the *i*th stock in the portfolio on the date `date`.

Figure 5: The contract for the `Portfolio` class, part 1.

*Public Class Methods:*

```
public static StockInfo stockFromLine (String line);
```

Returns a `StockInfo` instance for a line having the following form:

```
name (symbol) shares
```

where *name* is the name of the stock, *symbol* is the stock-ticker symbol of the stock, and *shares* is the number of shares of the stock.

```
public static Portfolio fromFile (String filename);
```

Creates a stock portfolio from the file named `filename`. The first line of this file should contain the name of the portfolio. Each remaining line of the file should contain a stock specification having the form of the line expected by `stockFromLine`. For example, the `ps2` directory contains a portfolio file `technology.txt` with the following contents:

```
Technology
Google (GOOG) 50
Apple Computer (AAPL) 150
Dell, Inc. (DELL) 200
Gateway, Inc. (GW) 1000
Microsoft (MSFT) 500
Oracle Corp. (ORCL) 200
Hewlett Packard (HPQ) 100
```

```
public static void main (String[] args);
```

There should be one element in `args`: the name of a file specifying a stock portfolio – e.g., `java Portfolio technology.txt`. When invoked in this manner, the `main()` method should (1) create a `Portfolio` instance *port* from the file (displaying a message as it reads the stock information for each stock), (2) display `port.toString()`; and (3) enter a read-eval-print loop (REPL) that allows the user to interactively test *port*.

In a REPL, the prompt should be the name of *port* followed by `>` and a space. E.g., the prompt for the portfolio created from `technology.txt` is `"Technology> "`. This REPL understands four kinds of inputs:

1. If the input is a single date *date* (written in the standard `CS230Date` format) that is in the range of quotes known by *info*, the output is the string returned by `verboseValueString()` for that date. If *date* is outside the known range of dates, an error message is displayed.
2. If the input is a pair of space-separated dates *start* and *stop* that are in the date range of the portfolio, the output is the value of the portfolio (one value per line) for *all* dates between *start* and *stop*, even ones for which individual quotes do not exist.
3. If the input has the form `extremes start stop`, the output is a verbose display of the portfolio for (1) the date in the range `[start,stop]` on which the portfolio value was a minimum; and (2) the date in the range `[start,stop]` on which the portfolio value was a maximum. If one or both of *start* and *stop* are not in the range of the portfolio, or if *start* > *stop*, then an error message is displayed.
4. If the input is `quit`, the REPL is terminated.

If the input is not one of the above forms, or if the dates specified are not in the right format, then a help message indicating the acceptable forms of input is displayed.

See Appendix B for a lengthy transcript of a sample REPL for `Portfolio`.

Figure 6: The contract for the `Portfolio` class, part 2.

## Appendix A: CS230Date Contract

The `CS230Date` class is a simple class used for representing dates consisting of a day, month, and a year for years between 0 and 9999. The class is named `CS230Date` to distinguish it from Java's standard `Date` class, which has many more features but is also more complicated to use.

*Public Class Variables:*

**public static final** `CS230Date` `MIN_DATE`;  
The minimal representable date = Jan. 1, 0000.

**public static final** `CS230Date` `MAX_DATE`;  
The maximal representable date = Dec. 31, 9999.

*Public Constructor Methods:*

**public** `CS230Date` (`int` year, `int` month, `int` day);  
Returns a date representing the given year (in the range 0–9999), month (in the range 1=January to 12=December), and day (in the range 1–31). Throws a `RuntimeException` for invalid dates (out-of-range years, months, or days, as well as Feb 29 in non-leap years, Feb 30, Feb 31, April 31, June 31, Sept 31, and November 31).

*Public Instance Methods:*

**public** `int` day ();  
Returns the day of this date (an integer in the range 0–31).

**public** `int` month ();  
Returns an integer specifying the month of this date (an integer in the range 1=January–12=December).

**public** `int` year ();  
Returns an integer specifying the year of this date (an integer in the range 0–9999).

**public** `CS230Date` next ();  
Returns the date that comes after this one. Throws a `RuntimeException` for Dec. 31, 9999.

**public** `CS230Date` prev ();  
Returns the date that comes before this one. Throws a `RuntimeException` for Jan. 1, 0000.

**public** `String` toString ();  
Returns a string representation of this date having the form `YYYY-MM-DD`, where `YYYY` are four digits specifying the year (in the range 0000–9999), `MM` are two digits specifying the month (in the range 01–12), and `DD` are two digits specifying the day (in the range 01–31).

**public** `boolean` equals (`Object` x);  
Returns `true` if `x` is a `CS230Date` representing the same date as this date, and `false` otherwise.

**public** `int` compareTo (`CS230Date` x);  
Returns a negative number if this date comes before `x`, 0 if this date is the same as `x`, and a positive number if this date comes after `x`.

*Public Class Methods:*

**public static** CS230Date fromString (String s);

The string *s* must be of the form *YYYY-MM-DD*, where *YYYY* are four digits specifying the year (in the range 0000–9999), *MM* are two digits specifying the month (in the range 01–12), and *DD* are two digits specifying the day (in the range 01–31). Returns a CS230Date instances representing the specified date. Throws a RuntimeException if the string *s* is of the wrong form or the specified date is invalid.

**public static** boolean isDayValidForMonth (int day, int month);

Returns true if day (in the range 1–31) is a valid day in the given month (1=January–12=December), and false otherwise.

**public static** boolean isLeapYear (int year);

Returns true if year denotes a leap year. A year is a leap year if it is divisible by 4, except for years divisible by 100 but not 400.

**public static** CS230Date min (CS230Date d1, CS230Date d2);

Returns the earlier of the dates d1 and d2.

**public static** CS230Date max (CS230Date d1, CS230Date d2);

Returns the later of the dates d1 and d2.

## Appendix B: A Transcript of the Portfolio Program

```
[fturbak@irwin ps2] java Portfolio small.txt
Reading stock information for Walmart (WMT) ...done.
Reading stock information for Coca-Cola (KO) ...done.
Reading stock information for General Motors (GM) ...done.
[Portfolio: Small Portfolio (date range [1972-08-25, 2007-02-15]):
1000 shares of Walmart (WMT)
    500 shares of Coca-Cola (KO)
    200 shares of General Motors (GM)
]

Small Portfolio> 1972-08-25
Portfolio value on 1972-08-25 = 1577.0:
    1000 shares of WMT @ 0.05 = 50.0
    500 shares of KO @ 1.11 = 555.0
    200 shares of GM @ 4.86 = 972.0000000000001

Small Portfolio> 2007-02-15
Portfolio value on 2007-02-15 = 79573.0:
    1000 shares of WMT @ 48.36 = 48360.0
    500 shares of KO @ 47.85 = 23925.0
    200 shares of GM @ 36.44 = 7288.0

Small Portfolio> 2000-01-01
Portfolio value on 2000-01-01 = 101181.0:
    1000 shares of WMT @ 65.17 = 65170.0
    500 shares of KO @ 50.81 = 25405.0
    200 shares of GM @ 53.03 = 10606.0

Small Portfolio> 01/01/2000
Type one date (YYYY-MM-DD) in the range [1972-08-25, 2007-02-15]
to get the value of the portfolio on that date.
Type two dates to get a listing of all portfolio values those dates.
Type extremes <start-date> <stop-date> to get the minimum/maximum portfolio values
```

between <start-date> and <stop-date>.  
Type quit to exit this read-eval-print loop.

```
Small Portfolio> 2007-01-07 2007-01-18
Portfolio value on 2007-01-07 = 77526.0
Portfolio value on 2007-01-08 = 77359.0
Portfolio value on 2007-01-09 = 77807.0
Portfolio value on 2007-01-10 = 77700.0
Portfolio value on 2007-01-11 = 78100.0
Portfolio value on 2007-01-12 = 78363.0
Portfolio value on 2007-01-13 = 78363.0
Portfolio value on 2007-01-14 = 78363.0
Portfolio value on 2007-01-15 = 78363.0
Portfolio value on 2007-01-16 = 78688.0
Portfolio value on 2007-01-17 = 78632.0
Portfolio value on 2007-01-18 = 78683.0
```

```
Small Portfolio> extremes 2007-01-07 2007-01-18
MINIMUM: Portfolio value on 2007-01-08 = 77359.0:
    1000 shares of WMT @ 47.0 = 47000.0
    500 shares of KO @ 48.57 = 24285.0
    200 shares of GM @ 30.37 = 6074.0
MAXIMUM: Portfolio value on 2007-01-16 = 78688.0:
    1000 shares of WMT @ 48.31 = 48310.0
    500 shares of KO @ 48.5 = 24250.0
    200 shares of GM @ 30.64 = 6128.0
```

```
Small Portfolio> extremes 1972-08-25 2007-02-15
MINIMUM: Portfolio value on 1974-12-06 = 637.0:
    1000 shares of WMT @ 0.01 = 10.0
    500 shares of KO @ 0.39 = 195.0
    200 shares of GM @ 2.16 = 432.0
MAXIMUM: Portfolio value on 2000-01-18 = 102272.0:
    1000 shares of WMT @ 61.81 = 61810.0
    500 shares of KO @ 56.7 = 28350.0
    200 shares of GM @ 60.56 = 12112.0
```

```
Small Portfolio> extremes 2007-02-15 1972-08-25
Invalid date range
```

```
Small Portfolio> extremes 2007-02-01 2007-02-28
Invalid date range
```

```
Small Portfolio> quit
```

```
[fturbak@irwin ps2] java Portfolio technology.txt
Reading stock information for Google (GOOG) ...done.
Reading stock information for Apple Computer (AAPL) ...done.
Reading stock information for Dell, Inc. (DELL) ...done.
Reading stock information for Gateway, Inc. (GW) ...done.
Reading stock information for Microsoft (MSFT) ...done.
Reading stock information for Oracle Corp. (ORCL) ...done.
Reading stock information for Hewlett Packard (HPQ) ...done.
[Portfolio: Technology (date range [2004-08-19, 2007-02-15]):
    50 shares of Google (GOOG)
    150 shares of Apple Computer (AAPL)
    200 shares of Dell, Inc. (DELL)
    1000 shares of Gateway, Inc. (GW)
    500 shares of Microsoft (MSFT)
    200 shares of Oracle Corp. (ORCL)
    100 shares of Hewlett Packard (HPQ)
]
```

```

Technology> extremes 2004-08-19 2007-02-15
MINIMUM: Portfolio value on 2004-08-19 = 33826.5:
    50 shares of GOOG @ 100.34 = 5017.0
    150 shares of AAPL @ 15.35 = 2302.5
    200 shares of DELL @ 34.7 = 6940.000000000001
    1000 shares of GW @ 3.99 = 3990.0
    500 shares of MSFT @ 23.55 = 11775.0
    200 shares of ORCL @ 10.42 = 2084.0
    100 shares of HPQ @ 17.18 = 1718.0
MAXIMUM: Portfolio value on 2007-01-16 = 74876.0:
    50 shares of GOOG @ 504.28 = 25214.0
    150 shares of AAPL @ 97.1 = 14565.0
    200 shares of DELL @ 26.51 = 5302.0
    1000 shares of GW @ 6.49 = 6490.0
    500 shares of MSFT @ 31.05 = 15525.0
    200 shares of ORCL @ 17.3 = 3460.0
    100 shares of HPQ @ 43.2 = 4320.0

```

```
Technology> quit
```

```

[fturbak@irwin ps2] java Portfolio bluechip.txt
Reading stock information for Procter & Gamble (PG) ...done.
Reading stock information for Merck (MRK) ...done.
Reading stock information for General Electric (GE) ...done.
Reading stock information for General Motors (GM) ...done.
Reading stock information for AT&T (T) ...done.
Reading stock information for Coca-Cola (KO) ...done.
Reading stock information for Disney (DIS) ...done.
Reading stock information for Exxon Mobil (XOM) ...done.
Reading stock information for Citigroup (C) ...done.
[Portfolio: BlueChip Stocks (date range [1984-07-19, 2007-02-15]):
    100 shares of Procter & Gamble (PG)
    100 shares of Merck (MRK)
    250 shares of General Electric (GE)
    500 shares of General Motors (GM)
    150 shares of AT&T (T)
    100 shares of Coca-Cola (KO)
    100 shares of Disney (DIS)
    150 shares of Exxon Mobil (XOM)
    100 shares of Citigroup (C)
]

```

```

BlueChip Stocks> extremes 1984-07-19 2007-02-15
MINIMUM: Portfolio value on 1984-07-24 = 6166.0:
    100 shares of PG @ 1.92 = 192.0
    100 shares of MRK @ 1.18 = 118.0
    250 shares of GE @ 1.14 = 285.0
    500 shares of GM @ 9.23 = 4615.0
    150 shares of T @ 1.87 = 280.5
    100 shares of KO @ 1.55 = 155.0
    100 shares of DIS @ 0.92 = 92.0
    150 shares of XOM @ 2.17 = 325.5
    100 shares of C @ 1.03 = 103.0
MAXIMUM: Portfolio value on 2000-04-26 = 75554.0:
    100 shares of PG @ 26.34 = 2634.0
    100 shares of MRK @ 54.93 = 5493.0
    250 shares of GE @ 46.55 = 11637.5
    500 shares of GM @ 67.55 = 33775.0
    150 shares of T @ 33.1 = 4965.0
    100 shares of KO @ 43.29 = 4329.0
    100 shares of DIS @ 39.6 = 3960.0

```

150 shares of XOM @ 34.23 = 5134.499999999999  
100 shares of C @ 36.26 = 3626.0

BlueChip Stocks> quit

[fturbak@irwin ps2]

*Problem Set Header Page*  
*Please make this the first page of your hardcopy submission.*

## CS230 Problem Set 2

### Due 6:00pm Friday February 23

Names of Team Members:

Date & Time Submitted:

Collaborators (*anyone you or your team collaborated with*):

*By signing below, I/we attest that I/we have followed the collaboration policy as specified in the Course Information handout.*

Signature(s):

*In the **Time** column, please estimate the time you or your team spent on the parts of this problem set. Team members should be working closely together, so it will be assumed that the time reported is the time for each team member. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the **Score** column when grading your problem set.*

| <b>Part</b>     | <b>Time</b> | <b>Score</b> |
|-----------------|-------------|--------------|
| General Reading |             |              |
| Problem 1 [20]  |             |              |
| Problem 2 [40]  |             |              |
| Problem 3 [40]  |             |              |
| <b>Total</b>    |             |              |