GEDIVA Team 1 API Document

Desired Features

- Simple chart view of the stock
    - Stock name
    - Latest Stock Price
- Trendline Drawing Tools
    - 50-day moving average
    - Simple linear trend lines
    - Channel (just two lines drawn from two price minimums and maximums)
- Interactive Chart
    - Adjustable dates
    - Activating and deactivating view of analysis tools

Design Scheme

The model view controller paradigm will be used throughout this design. The model serves as the wrapper that holds all the data known to the system. It is responsible for building, processing, and returning requested data. The controller is the middleman between the model and view. It serves as the glue logic that allows the views to interact with the model. The views are responsible for displaying and receiving information from the user.

Data is passed back and forth from the controller and the model via DataSet objects and a single request method. To retrieve data from the model, a request is called by using the desired supported RequestType as an argument. This request is then passed onto the model, and in response, the model builds a DataSet object. This object once again contains all the relevant information that came about from the request. This scheme was chosen because it does not limit the range of available requests, and it helps the view by giving the response data in one object that also has some functionality and adaptability. Doing so allows us to not have random strings, lists, maps, and such, floating around.

Analogy

We are using MVC, which is like a restaurant. A customer (view) just makes requests and is served by the waiter (controller) who forwards those requests to the chef (model).

Class Listing and Hierarchies

Controller

- Controller (Abstract) - initiates model & views and manages all interaction
    - StockController - adds stock specific methods

View

- Canvas - holds the views, instigates painting, and propogates mouse clicks
- View - displays an element on the canvas - manages children and mouse clicks

    - Graph (Abstract) - paints given data into a graph
        - BarGraph - bar variant - implements painting the data
        - LineGraph - line variant - implements painting the data
    - Button - represents a clickable element
    - ErrorView - used to display errors to the user (like a failure to load a file)
    - Header - represents a title/header for a visualization - has some stock specific properties
    - Menu - holds a set of buttons
- StockViewFactory - generates the views used for stock data visualization

Model

- AbstractModel (Abstract) - can populate its dataTable and process requests
    - StockModel - implements methods and adds stock related info

Data

- DataTable (Abstract) - internals for model - has content population and sorting methods
    - StockTable - implements methods for stocks
- RowElement (Abstract) - component of DataTable - can intelligently compare with other RowElements
    - StockRowElement - implements methods for stocks
- IDataSet (Interface) - returned by requests to model - is a wrapper for a DataTable
    - StockDataSet - interfaces with StockTable

Parsers

- GenericParser (Abstract) - factory that can turn an input (designated by filename, url, etc.) into a BufferedReader
    - WebParser - can read from a socket
    - FileParser - can read from a file

Utility

- Date - a simple class which is capable of formatting dates - easier to use than Java's built in date and calendar

Complete Public Method Listing

Controller

```
/**
 * Abstract controller class that defines
 * some methods to be used by a subclass
 * of controller specific to a type of data.
 */
public abstract class Controller {
    /**
     * Initializes the controller with a canvas.
     *
```

```
     * @param display is the canvas (screen)
     */
    public void init (Canvas display)

    /**
     * Lets the user choose a file to download
     * data.
     */
    public void chooseFile ()

    /**
     * Lets the user type in a URL to load in data from.
     */
    public void chooseUrl ()

    /**
     * Toggles the type of graph on the screen between
     * line or bar graph.
     */
    public void toggleGraph ()
}
```

StockController

```
/**
 * The controller that is used when downloading
 * stock data.
 */
public class StockController extends Controller {
    /**
     * Initializes the stock controller by
     * calling the initializer for the
     * superclass.
     *
     * @param c is the canvas from which
     *        this controller is initialized.
     */
    public StockController (Canvas c)


    /**
     * Lets the user choose a source URL by the symbol
     * of a stock that they want.
     */
    public void chooseUrlBySymbol ()

    /**
     * Generates the view's menu and puts it in the canvas
     */
    public void startMenu ()
```

```
    // button responders....

    /**
     * functionality for the "Load from Symbol" button
     *
     * @param attributes Button attributes (unused for this function)
     */
    public void respondToChooseSymbol (HashMap<String, String> attributes)

    /**
     * functionality for the "Load from URL" button
     *
     * @param attributes Button attributes (unused for this function)
     */
    public void respondToChooseUrl (HashMap<String, String> attributes)

    /**
     * functionality for the "Load File" button
     *
     * @param attributes Button attributes (unused for this function)
     */
    public void respondToChooseFile (HashMap<String, String> attributes)

    /**
     * functionality for the "Switch graph view" button
     *
     * @param attributes Button attributes (unused for this function)
     */
    public void respondToToggleGraph (HashMap<String, String> attributes)
}
```

Canvas

```
/**
 * This class describes how to display/initialize
 * the canvas (window screen displayed to the user)
 * for the current program.
 */
public class Canvas extends JComponent {
    /**
     * Initializes the canvas (window screen)
     * to be displayed to the user.
     *
     * @param size of the canvas
     */
    public Canvas (Dimension size)

    @Override
    public void paint(Graphics g)

    /**
```

```
     * Adds a view (graph or label) to the root view.
     *
     * @param v the view to be added
     */
    public void addView(View v)

    /**
     * Updates all of the views of this canvas.
     */
    public void update()

    /**
     * @return the root view
     */
    public View getRoot()
}
```

View

```
/**
 * This class describes how to paint a single view and/or
 * its children. The root view (in canvas) is actually the only
 * paint method called explicitly by the program, and that root view calls
 * paint on all it's children recursively.
 */
public class View extends JComponent {
    /**
     * Initializes a view.
     *
     * @param position of the top left corner
     * @param size of the view
     * @param bgcolor background color
     */
    public View(Point2D position, Dimension size, Color bgcolor)

    /**
     * Initializes a view.
     *
     * @param position of the top left corner
     * @param size of the view
     */
    public View(Point2D position, Dimension size)

    /**
     * Paints the view and all of its children.
     *
     * @param pen used to paint
     */
    public void paint(Graphics2D pen)

    /**
```

```
 * Adds a child to this view.
 * The children are graphs or labels (currently).
 *
 * @param v the view to be added
 */
public void addChild(View v)

/**
 * Removes a child from this view
 * (in the case of deletion/editing of a graph
 * or reloading of data).
 *
 * @param v the view to be removed
 */
public void removeChild (View v)

/**
 * Offsets the position of the view so that
 * it and its parent do not overlap.
 *
 * @param offset the point to offset this view's
 *        position to
 */
public void offsetPosition (Point2D offset)

/**
 * Returns the type of view.
 */
public String getType ()

/**
 * Decides where the mouse is clicked and determines
 * whether the click is handled by this view or
 * one of its children.
 *
 * @param point of the mouse click
 */
public void mouseClicked (Point point)

/**
 * Returns the position of this view.
 */
public Point2D getPosition ()

/**
 * Returns the size of the view.
 */
public Dimension getSize ()

/**
```

```
     * @return the list of children of this view
     */
    public List<View> getChildren ()

    /**
     * Brings up a file dialogue box that allows the user to choose a
     * file from the filesystem
     *
     * @param canvas to display error if something goes wrong with file
selection
     * @return chosen file (may be null if there is an error with file
selection)
     */
    public static File chooseFile (Canvas canvas)

    /**
     * Toggles the type of a displayed graph
     *
     * @param canvas container of view to be toggled
     */
    public static void toggleGraph (Canvas canvas)

    /**
     * Displays an error message in the given canvas
     *
     * @param canvas container in which to show the error
     * @param message error message to display
     */
    public static void showError (Canvas canvas, String message)
}
```

Graph

```
/**
 * Generic graph class, used to make
 * graphs (x and y axis types) for sets of data. Graph takes
 * a position (from which to draw the background),
 * a size of the background, a Map of x axis values
 * to y axis values, and labels for x axis and y axis.
 * This class deals with setting up the basic information
 * for the graph and painting the axes, labels,
 * and values on the axes. Painting the graph's
 * points, bars, lines, etc. is up to the subclass
 * of graph since there are different kinds of
 * ways to show (paint) the data.
 *
 *
 * @param <K> the type of xValue (usually dates)
 * @param <V> the type of yValue (e.g. price)
 */
public abstract class Graph<K extends Comparable<? super K>,
```

```
        V extends Comparable<? super V>> extends View {

    /**
     * Initializes the graph's origin, x-axis length/label,
     * y-axis height/label, and x and y values.
     *
     * @param position the position of the top left corner
     * of the graph window
     *
     * @param size of the window of the graph
     * @param values a map of x values to corresponding
     * y values
     *
     * @param x the label on the x axis
     * @param y the label on the y axis
     */
    public Graph (Point2D position, Dimension size, Map<K, V> values,
            String x, String y)

    /**
     * Paints background, axis, and labels of the graph.
     *
     * @param pen is used to paint
     */
    @Override
    public void paint(Graphics2D pen)

    /**
     * Returns the type of graph.
     */
    public abstract String getType()

    /**
     * @return the map of x values to y values
     */
    public Map<K, V> getVals()

    /**
     * Returns the point of the
     * origin of the graph (relative
     * to the Canvas origin).
     *
     * @return
     */
    public Point2D getMyOrigin ()
}
```

BarGraph

```
/**
 * Specifies the specifics for painting a bar graph.
```

```
 */
public class BarGraph extends Graph<Date, Double> {
    /**
     * Initializes a bar graph from passed information.
     *
     * @param position the position of the top left corner
     * of the graph window
     *
     * @param size of the window of the graph
     * @param values a map of x values to corresponding
     * y values
     *
     * @param xAxisLabel the label on the x axis
     * @param yAxisLabel the label on the y axis
     */
    public BarGraph (Point2D position, Dimension size,
            Map<Date, Double> values, String xAxisLabel, String yAxisLabel)

    /**
     * Paints the bars corresponding
     * to the data held by the graph.
     *
     * @param pen used to paint
     */
    public void paintData(Graphics2D pen)

    /**
     * Returns the type of this graph (bar).
     */
    @Override
    public String getType()
}
```

LineGraph

```
/**
 * Creates and draws a line graph for the data in Map<> values.
 * Uses the abstract class Graph to determine its points, and
 * uses its own paintGraph method to paint the data plot info.
 */
public class LineGraph extends Graph<Date, Double> {

    /**
     * Initializes a line graph from given information.
     *
     * @param position the position of the top left corner
     *        of the graph window
     *
     * @param size of the window of the graph
     * @param values a map of x values to corresponding
     *        y values
```

```
     *
     * @param xAxisLabel the label on the x axis
     * @param yAxisLabel the label on the y axis
     */
    public LineGraph (Point2D position, Dimension size,
            Map<Date, Double> values, String xAxisLabel, String yAxisLabel)

    /**
     * Paints the points and the line connecting them.
     *
     * @param pen is used to draw
     */
    public void paintData(Graphics2D pen)

    /**
     * Returns the type of this graph (line).
     */
    public String getType()
}
```

Button

```
/**
 * This class creates the buttons displayed
 * on the side of the window.
 */
public class Button extends View {
    /**
     * Initializes a button's position, size and label.
     *
     * @param position of the top left corner of the button
     * @param size of the button
     * @param message displayed on the button
     */
    public Button(Point2D position, Dimension size, String message)

    /**
     * Initializes a button, but with background and text colors.
     *
     * @param position of the top left corner of the button
     * @param size of the button
     * @param message displayed on the button
     * @param bgColor background color
     * @param textColor text color
     */
    public Button(Point2D position, Dimension size,
            String message, Color bgColor, Color textColor)

    /**
     * Paints the button on the screen.
     *
```

```
     * @param pen used to paint
     */
    public void paint(Graphics2D pen)

    /**
     * Adds attributes to this button (determines what to do).
     *
     * @param key the action
     * @param value the result
     */
    public void addAttribute (String key, String value)

    /**
     * Sets the method for this button.
     * The method m must be a method of the
     * controller c.
     *
     * @param m the name of the method to set
     * @param c the controller of the button
     */
    public void setMethod(String m, StockController c)

    /**
     * Set the instance that will respond to clicks.
     *
     * @param c the controller of the button
     */
    public void setResponder (Controller c)

    @Override
    public void mouseClicked(Point p)
}
```

ErrorView

```
/**
 * This class specifies how to display an
 * error on the canvas.
 */
public class ErrorView extends View {
    /**
     * Initializes an error message to be displayed on the screen.
     *
     * @param position of the top left corner of the error message
     * @param size of the error message label
     * @param message displayed on the label
     * @param bgColor background color
     * @param textColor text color
     */
    public ErrorView(Point2D position, Dimension size,
            String message, Color bgColor, Color textColor)
```

```
    /**
     * Initializes an error message to be displayed on the screen.
     *
     * @param position of the top left corner of the error message
     * @param size of the error message label
     * @param message displayed on the label
     */
    public ErrorView(Point2D position, Dimension size, String message)

    @Override
    public void paint(Graphics2D pen)
}
```

Header

```
/**
 * Creates the header for the current view screen.
 * Dependent on current stock that is loaded.
 */
public class Header extends View {
    /**
     * Initializes the header for the currently loaded stock on the canvas.
     *
     * @param position of the top left corner of the header
     * @param size of the header label
     * @param title main title of the header
     * @param subtitle of the header
     * @param price the price of the currently displayed stock
     * @param bgcolor background color
     * @param wordColor text color
     */
    public Header (Point2D position, Dimension size, String title,
            String subtitle, String price, Color bgcolor, Color wordColor)

    /**
     * Initializes the header for the currently loaded stock on the canvas.
     *
     * @param position of the top left corner of the header
     * @param size of the header label
     * @param title main title of the header
     * @param subtitle of the header
     * @param price the price of the currently displayed stock
     */
    public Header (Point2D position, Dimension size, String title,
            String subtitle, String price)

    /**
     * Initializes a generic header.
     *
     * @param position of the top left corner of the header
```

```
     * @param size of the header label
     * @param title main title of the header
     * @param subtitle of the header
     */
    public Header (Point2D position, Dimension size, String title,
            String subtitle)

    /**
     * Changes the title of this header.
     *
     * @param title what to change the main
     * title to
     */
    public void setTitle(String title)

    /**
     * Changes the subtitle of this header.
     *
     * @param subtitle the string to change
     * the subtitle to
     */
    public void setSubtitle(String subtitle)

    /**
     * Changes the price displayed by this header.
     *
     * @param price the new price
     */
    public void setPrice(String price)

    @Override
    public void paint(Graphics2D pen)
}
```

Menu

```
/**
 * This class describes how to paint the menu
 * on the canvas.
 */
public class Menu extends View {
    /**
     * Initializes the menu to be painted.
     *
     * @param position of the top left corner
     * @param size of the menu
     * @param title of the menu
     * @param bgColor background color
     * @param textColor text color
     */
    public Menu (Point2D position, Dimension size, String title,
```

```
                Color bgColor, Color textColor)

    /**
     * Initializes the menu to be painted.
     *
     * @param position of the top left corner
     * @param size of the menu
     * @param title of the menu
     */
    public Menu (Point2D position, Dimension size, String title)

    @Override
    public void paint (Graphics2D pen)
}
```

StockViewFactory

```
/**
 * In charge of populating a canvas on behalf of a controller
 * Specific to stocks
 */
public class StockViewFactory {
    /**
     * Constructor specific to a controller (listens to events)
     * and a canvas (holds generated the views)
     *
     * @param controller in charge of event handling
     * @param canvas to populate
     */
    public StockViewFactory (StockController controller, Canvas canvas)

    /**
     * Generates the view's menu and puts it in the canvas
     */
    public void startMenu ()

    /**
     * Populates a line graph with data and puts it and a header onto the
canvas
     *
     * @param dataSet contains the data to be used in the graph
     * @param info contains the info to put in the header
     */
    public void startCanvas (IDataSet<Comparable> dataSet,
                             Map<String, String> info)
}
```

AbstractModel

```
/**
 * Describes the abstract implementation for the
 * model used by the program.
```

```
 */
public abstract class AbstractModel {
    /**
     * Initializes a model for use by the program.
     */
    public AbstractModel()

    /**
     * Returns an unmodifiable list of supported parsers.
     * @return
     */
    public List<String> supportedParsers()

    /**
     * Initializer expects a Scanner. It calls helper methods to parse the
data
     * and store it in the appropriate data structure(s).
     *
     * @param name of the source from which to get the raw data
     */
    public boolean initialize (String name)

    /**
     * Returns true/false depending on the success
     * of initializing from a file.
     *
     * @param file from which to try initializing
     * @return
     */
    public boolean initialize(File file)

    /**
     * Gets the identifier of this model.
     */
    public abstract String getIdentifier ();

    /**
     * this is a helper method to parse data. It can be called explicitly
     * outside of init in case you wanted to reload the data or load different
     * data.
     *
     * @param s the source from which to load data
     */
    public abstract boolean load (BufferedReader s);

    /**
     * receives a Request object. The request contains the request it should
     * respond to.
     *
     * @param requestType the type of data requested
```

```
     */
    public abstract IDataSet<?> process (String requestType);

    /**
     * Returns a list of possible request types.
     */
    public abstract String[] getRequestTypes ();
}
```

StockModel

```
/**
 * This class holds all the info and describes how to
 * use/modify it for a current stock.
 */
public class StockModel extends AbstractModel {
    /**
     * Initializes a model specific to stock data.
     *
     * @param symbol the ticker symbol of the stock
     * @param companyName the name of the company
     */
    public StockModel (String symbol, String companyName)

    /**
     * Parses the data and performs some stock specific parsing, like
extracting
     * the name and ticker symbol.
     *
     * @param s the source from which to load
     */
    @Override
    public boolean load (BufferedReader s)

    /**
     * Returns basic info about the stockÔøΩ name, ticker symbol, etc.
     */
    public Map<String, String> getStockInfo ()

    /**
     * Sets the symbol and stock name.
     *
     * @param symbol the ticker symbol of the stock
     * @param name of the stock
     */
    public void setStockInfo (String symbol, String name)

    @Override
    public String getIdentifier ()

    @Override
```

```
    public IDataSet<Comparable> process (String requestType)

    @Override
    public String[] getRequestTypes ()
}
```

DataTable

```
/**
 * Abstract class for initializing the
 * data table used by the rest of the
 * program.
 *
 * @param <T> is the type of data represented
 *        in the table.
 */
public abstract class DataTable<T> {
    /**
     * Constructor for the data table.
     * Simply initializes the rows and
     * columns as empty arraylists.
     */
    public DataTable ()

    /**
     * Initializes this data table based on
     * one that is passed in.
     *
     * @param startFrom is the data table from which to
     *        initialize the current data table
     */
    public DataTable (DataTable<T> startFrom)

    /**
     * Sets the column names of this data table.
     *
     * @param s is the string holding the column
     *        names (all separated by commas).
     */
    public void setColumnNames (String s)

    /**
     * Setting a column as the primary key.
     *
     * @param colname is the name of the column
     *        from which to set the primary key.
     */
    public void setPrimaryKey (String colname)

    /**
     * Sorts the data table by column.
```

```
 *
 * @param colname is the name of the column
 *        from which to base the sorting procedure.
 */
public void sortbyColumn (String colname)

/**
 * Adds an empty column to the data table.
 *
 * @param colname the name of the column
 *        to be added.
 */
public void addColumn (String colname)

/**
 * Sets the values for one column (and all RowElements)
 *
 * @param colname name on the column to set
 * @param it iterator with values to be set (in the same order as the
 *        corresponding rows)
 */
public void setColumnValues (String colname, Iterator<T> it)

/**
 * Clears the table of all data.
 */
public void clear ()

/**
 * Reverses the rows of the data table.
 */
public void reverse ()

/**
 * Returns an unmodifialbe version of the
 * column names for this table.
 */
public List<String> columnNames ()

/**
 * Returns the values of one column in an unmodifiable list (in the same
 * order as the current rows)
 *
 * @param attribute name of the column to get data from
 * @return unmodifiable list of the specified column's data
 */
public List<T> columnValues (String attribute)

/**
 * Removes the row specified by index.
```

```
     *
     * @param index the position of the row
     *        to be removed (from the top).
     */
    public void removeRow (int index)
}
```

StockTable

```
/**
 * Specifies an instance of DataTable for implementing
 * stock data.
 */
public class StockTable extends DataTable<Comparable> {
    /**
     * Initializes and empty data table.
     */
    public StockTable ()

    /**
     * Initializes a data table for stock info.
     *
     * @param st is the data table from which to
     *        initialize.
     */
    public StockTable (DataTable<Comparable> st)

    /**
     * Add a new row to this data table.
     *
     * @param sRow the row to add to the data
     *        table (info is separated by commas).
     */
    public void newRow (String sRow)
}
```

RowElement

```
/**
 * Provides an abstract representation of an elemnt of data
 * contained within a row of the DataTable.
 *
 * @param <T>
 */
public abstract class RowElement <T> implements Comparable<RowElement <T>> {
    @Override
    public abstract int compareTo (RowElement<T> r);
}
```

StockRowElement

```
/**
 * This class encompasses the information needed to
```

```
 * store and modify elements of a row of stock data.
 */
public class StockRowElement extends RowElement<Comparable> {
    /**
     * Initializes an element of a row of stock data.
     */
    public StockRowElement ()

    @Override
    public int compareTo (RowElement<Comparable> r)

    /**
     * Adds another element to this row of stock data.
     *
     * @param rdata the data to add to this row
     */
    public void addData (Comparable rdata)
}
```

IDataSet

```
/**
 * Provides an interface for a set
 * of data.
 *
 * @param <T> the type of data in the
 * data set.
 */
public interface IDataSet<T> {

    /**
     * Gets the data according to an attribute.
     *
     * @param attribute what to look for
     *        when getting the data.
     * @return a list containing ordered
     *        data of the specified attribute
     */
    List<T> getData(String attribute);

    /**
     * Sorts the data in the data set
     * according to an attribute.
     *
     * @param attribute what to sort
     *        the data by.
     * @return a sorted version of the dataSet
     */
    IDataSet<T> sort(String attribute);

    /**
```

```
     * Returns a list of attributes held
     * by the data table.
     *
     * @return list of attributes
     */
    List<String> attributes();
}
```

StockDataSet

```
/**
 * Instance of IDataSet that is specific to stock data.
 */
public class StockDataSet implements IDataSet<Comparable> {
    /**
     * Initializes this stock data set.
     *
     * @param st the data table that holds the data
     * for this data set.
     */
    public StockDataSet (DataTable<Comparable> st)

    @Override
    public List<Comparable> getData (String attribute)

    @Override
    public StockDataSet sort (String attribute)

    @Override
    public List<String> attributes ()
}
```

GenericParser

```
/**
 * Abstract class for a file parser from which
 * to get the data.
 */
public abstract class GenericParser {
    /**
     * Creates a generic parser that is empty.
     */
    public GenericParser()

    /**
     * Generates a parser to read from a certain file.
     *
     * @param name is the name of the file from which
     * to read
     *
     * @return BufferedReader - parser from a specified input
     * @throws IOException for any other error
```

```
     */
    public BufferedReader generateReader(String name) throws IOException

    /**
     * Determines whether the file is readable/openable.
     *
     * @param name is the name of the file
     * @return the reader can support this input type
     */
    public abstract Boolean isSupported(String name);
}
```

FileParser

```
/**
 * Creates a parser that takes information from
 * a file.
 */
public class FileParser extends GenericParser {
    /**
     * Initializes the parser to read from a file.
     *
     * @param file the file from which to read
     * @return Reader - can read from specified file
     * @throws IOException the exception to be thrown
     * if the file is invalid
     */
    public Reader getReader(File file) throws IOException

    /**
     * Returns whether the file can be read or not.
     *
     * @param name is the string name of a file
     */
    public Boolean isSupported (String name)

    /**
     * Returns whether the file can be read or not.
     *
     * @param file is the file from which to read
     */
    public Boolean isSupported(File file)
}
```

WebParser

```
/**
 * Creates a parser that takes information from
 * the web.
 */
public class WebParser extends GenericParser {
    /**
```

```
     * Returns whether the parser is workable.
     *
     * @param name is the name of the source of
     * the web parser.
     */
    public Boolean isSupported (String name)
}
```

Date

```
/**
 * A basic utility class for storing dates in a readable
 * and manipulatable format.
 */
public class Date implements Comparable<Date> {
    /**
     * Initializes a date if the info is in number format.
     *
     * @param year of the date
     * @param month of the date
     * @param day of the date
     */
    public Date (int year, int month, int day)

    /**
     * Initializes a date if the input is in string format.
     *
     * @param year of the date
     * @param month of the date
     * @param day of the date
     */
    public Date (String year, String month, String day)

    /**
     * Prints the short format of this date.
     */
    public String toString ()

    /**
     * Returns the long format of the date.
     */
    public String getLongFormat ()

    /**
     * @return the myYear
     */
    public int getYear ()

    /**
     * @param year the year to set
     */
```

```
    public void setYear (int year)

    /**
     * @return the myMonth
     */
    public int getMonth ()

    /**
     * @param month the month to set
     */
    public void setMonth (int month)

    /**
     * @return the myDay
     */
    public int getDay ()

    /**
     * @param day the day to set
     */
    public void setDay (int day)

    /**
     * Sorts date by year, then month, then day.
     *
     * @param d the date to compare against
     */
    @Override
    public int compareTo (Date d)
}
```