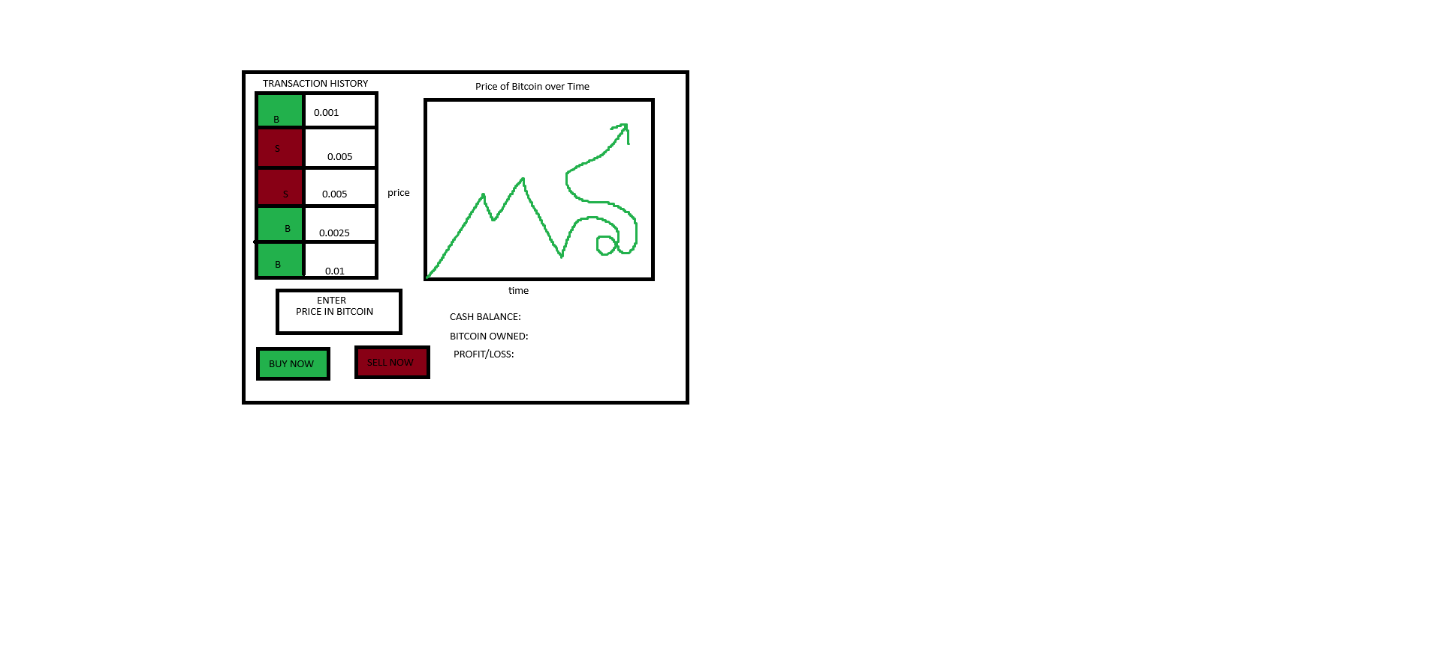
* An **overview** of your project and your motivation for choosing this topic and the way(s) in which your project creatively goes beyond what we have done in class and labs.
  + Bitcoin Simulator
    - My project will be a bitcoin simulator where you can buy and sell bitcoin while it shows how much money you have left, your profit/loss, and tracks bitcoin over time on a chart/graph. The data for remembering how much fake money you have and how much you’ve made/lost will be stored in a file.
    - My motivation for choosing this topic is that I would like to learn how to make charts/graphs in javafx because it is something we didn’t really learn how to do. I also chose bitcoin because I can obtain live data for free (if I did stocks, etc. I would have to pay)
* A **description of the classes** you will use to decompose the information in your project, with a list of the variables and methods in each class.
  + BitcoinApp.java – This will help run the JavaFX GUI
  + Account.java – this is an account object that has a constructor that stores the cash balance (cashBalance), bitcoin balance (bitcoinBalance), total $ spent on bitcoin (totalBought), and total $ sold on bitcoin (totalSold)
    - Basic getter methods e.g., getCashBalance(), getBitcoinBalance(), getTransactionLog()
    - Basic setter methods e.g., setCashBalance(), setBitcoinBalance()
    - buyBitcoin(amount, currentPrice) – this method is called to buy bitcoin and it sets the account stats accordingly
    - sellBitcoin(amount, currentPrice) – this method is called to sell bitcoin and it sets the account stats accordingly
  + Transaction.java – this is a transaction object that has a constructor that stores the type of transaction (type), amount in bitcoin (amount), time of the transaction (date), the cash balance (cashBalance), bitcoin balance (bitcoinBalance), and price of bitcoin at the time a transaction was made, (bitcoinPrice)
    - Basic getter methods e.g., getType(), getAmount(), getDate(), getCashbalance(), getBitcoinBalance(), getBitcoinPrice()
  + TransactionLog.java – this helps keep track of and storing the transactions (Transaction.java objects) in a csv file (data.csv). It stores the type of transaction, date of transaction, amount (in bitcoin) of the transaction, cash balance left after processing it, the bitcoin balance after the transaction, and the price of bitcoin at the time the transaction was made. This helps with keeping the values the same when reloading the program so the data isn’t lost when you close it.
    - saveTransaction(transaction) – this method saves the transaction to the csv file
    - getAllTransactions() – this method returns a list of all the transactions in the csv file to copy to the transaction log on the GUI.
    - retrieveFromCsv(part) – this method is used to retrieve a specific part from the
    - isEmpy() – this checks to see if the csv file is empty or not
  + TransactionType.java (enum)
    - Basic enum with BUY & SELL to help differentiate between the types of orders being made.
  + API.java – this method uses HttpsURLConnection and Jackson typereference, jsonnode, and objet mapper to retrieve data from an api to get the price history of bitcoin and the current price of bitcoin
    - getBitcoinPrice() – returns the current price of bitcoin
    - getBTCPriceHistory(accuracy, daysBehind) – returns a TreeMap of dates/times of their corresponding prices. Accuracy is how often (hourly, daily, etc) and daysBehind is how long ago the start Time is.
  + BitcoinController.java – the main part of the program. It helps the user interact with the GUI and helps manage the account and its attributes, and the transaction history. It also communicates with the API class to get the live data and price history
    - Variables
      * Account account – is initialized in initialize() and creates the account object
      * int startingBalance – the starting balance of account, initially at 1000000
      * Movement timer – timer that helps with updating the current price of bitcoin on the screen(above the chart)
      * int days behind – the start time of the price history
      * API api – the api that helps retrieve data
      * float currentBitcoinPrice – current bitcoinPrice
      * TreeMap<LocalDateTime, Float> priceHistory; - price history of bitcoin
      * @FXML variables
        + LineChart<Number, Number> lineChart;
        + Button chartRefresh;
        + Text currentPriceText;
        + Text cashBalanceText;.
        + Text bitcoinBalanceText;
        + Text profitText;
        + TextField bitcoinTextField;
        + private Button buyButton;
        + Button sellButton;
        + Button buyAllButton;
        + Button sellAllButton;
        + TableView<Transaction> transactionLogTable;
        + TableView<Transaction> transactionLogTable;
        + TableColumn<Transaction, TransactionType> transactionLogTableType;
        + TableColumn<Transaction, Float> transactionLogTableAmount;
        + TableColumn<Transaction, String> transactionLogTableDate;
        + TableColumn<Transaction, Float> transactionLogTableCashBalance;
        + TableColumn<Transaction, Float> transactionLogTableBitcoinBalance;
    - Methods
      * Initialize() – loads the inital state of the application and brings the account data from the CSV and updates it. (so it isn’t lost when exiting the program)
      * Handle() – handles the intervals and how often the chart and price get updated by the API
      * @FXML methods
        + buyBitcoin()
        + buyAllBitcoin()
        + sellBitcoin()
        + sellAllBitcoin()
      * updateAccountStats()
      * updateChart()
      * updateCurrentBitcoinPrice()
      * getTotalProfit()
      * getHighestPrice()
      * getLowestPrice()
* A discussion of **the data structure you are incorporating**, why you selected it, and why the asymptotic time complexity of its operations is a good match for your project requirements.
  + I chose to do a TreeMap because they can be ordered by date which is what I want since I’m tracking the price of bitcoin over a certain period of time. It is also efficient because of its time complexity of O(log(n)). It also has key-value pairs which is useful for what I’ll be working on because the key would be the date/timestamp and the value would be the price of bitcoin.
* A **sketch** of the JavaFX GUI for your project.
  + 
* A **timeline** of steps you will take to complete your project with deadlines for each step. Plan not just what you must do but the ordering of steps that will allow you to complete by the final deadline.
  + April 16
    - Begin working on design
  + April 25
    - Turn in design
  + April 26-27
    - Work on GUI w/ scenebuilder
  + April 28-29
    - Work on classes & methods in application
  + April 30-May 1
    - Test out program & debug/check for errors
  + May 8
    - Present