DESIGN DOCUMENT
FOR
BITCOIN SIMPLE PAYMENT VERIFICATION CLIENT

February 21, 2017

Prepared by:

Frank Fasola
James Donnell
Spencer Escalante
Trevor Silva

Table of Contents

1 Purpose

This document presents the Solution Development Lifecycle (SDLC) design for the Senior Project affecting the Bitcoin Simple Payment Verification Client, Bitcoin SPV Client throughout.

2 Scope

This design document describes the system-wide design and architecture for the Bitcoin SPV Client. The document includes sections based on the Coding Standards, Solution Design Specification, Configuration Settings, Technical Architecture and initial Implementation Phase. This document will be updated during the Design Phase of the Bitcoin SPV Client and as the system design matures in order to accommodate any changes in the design.

2.1 Exclusions, Assumptions, and Limitations

The assumption is made that the Bitcoin SPV Client will be run on a Windows Operating System (OS). The computer must have read and write access to the logged-on user's application data (%appdata% → C:\Users\{USERNAME}\AppData\Roaming) folder. This access is default in most Windows environments.

The Bitcoin SPV Client is designed in a way where it is specifically not a full node, but rather a lightweight node.

The Bitcoin SPV Client is designed to run as a single instance on a single computer. Multiple instances of the client on a single computer may work unharmed, but the attempt to make such environments function is out of the scope of the Bitcoin SPV Client.

2.2 System Overview

The Bitcoin SPV Client is required to connect to peer Bitcoin client(s) to gather Blockchain header information, maintain wallet balance and provide simple payment verification. It should therefore allow users a way to view Blockchain information, view their wallets and verify payments.

3 Solution Design Overview

The Bitcoin SPV Client will emphasize a Client-Client model in a Peer-to-Peer (P2P) network typical in Bitcoin clients. The Client-Client model will allow the Bitcoin SPV Client to connect to multiple Bitcoin peers to increase the accuracy of the overall Blockchain data. The client will also emphasize it's functionality as a lightweight node. This meaning it does not download the entire Bitcoin Blockchain as full nodes, instead it only downloads the Blockchain headers initially. It can then request an individual full block as needed.

The solution will use a Model-View-Controller (MVC) design pattern to develop the client in a well-organized manner. MVC is a design pattern used to keep program code organized when designing user interfaces. It accomplishes this by splitting the overall project into three major components used. The Controller module, based on its name, controls the entire application. It determines how to process data and information from the Model module and determines what to show on the View module. The view module will be a simple Graphical User Interface (GUI) that communicates with the controller and in turn, the model in order to make sense of the Bitcoin Blockchain data for the users.

4 Technical Architecture

4.1 Hardware Inventory, Specifications and Locations

4.1.1 Computers

A single computer is needed for each Bitcoin SPV client that is desired to be running. Multiple client instances on a single computer may be possible, but is not considered in the scope of this design or project. At a minimum, each computer must have the following specifications:

- 1GB of free Hard Disk Drive/Solid State Drive space
- Mouse and/or Keyboard input
- Network connectivity to Bitcoin peers on Internet
- Windows OS
- Java 1.8 or higher

4.1.2 Input / Output Devices

Input to the SPV client takes two forms; the first is input from a user. Input from a user is received from a mouse or keyboard. The second input comes from other nodes on the P2P network. Nodes can send input in the form of block headers to be processed by the lightweight node.
Output from the system is the collection of block headers the lightweight node receives when it first connects to the bitcoin P2P network.

4.1.3 Other Devices

The Bitcoin SPV Client does not require any other devices.

4.1.4 Infrastructure / Application Diagram

The Bitcoin SPV Client does not require any complex network connections or components that require explicit description.

4.2 Interfaces with Other Hardware and External Integration Points

The Bitcoin SPV Client does not interface with other external hardware or integrations points.

4.3 Physical Layout

The Bitcoin SPV Client requires a computer to have an active network connection to a network which provides unblocked access to Bitcoin Peers on the internet.

5 Configuration Specifications

5.1 Installation

The user does not need to install software to their computer to use the Bitcoin client. The client will be offered as a pre-compiled executable file as a .EXE file. Users can download the executable file and launch the client, then the client will begin to download the block headers and connect to nodes on the Bitcoin network.

5.2 Configuration Settings

The Bitcoin SPV requires no special configurations to be made to run in a Windows environment. As the Bitcoin client is a pre-compiled executable file, there will not be settings for users to change to better suit their needs before the program's first launch.

6 Solution Design Specification

6.1 Software Description

The client will utilize the Model-View-Controller (MVC) design pattern as a basis for implementation.

The Bitcoin SPV Client will utilize libraries created by the open-source project called bitcoinj. This open-source project is a collection of Java classes that enable Java developers the ability to interact with Bitcoin Peers easily and efficiently to retrieve Bitcoin Blockchain data. Due to it's functionality, bitcoinj provides numerous classes and packages for the lifecycle of a Bitcoin client. This includes peer connection to Blockchain downloading to Blockchain model classes. Some of the most notable classes are NioClient.java and NioClientManager.java which specifically handle the network connection to individual peers and the downloading of the Blockchain.

6.2 Coding Standards

The Bitcoin SPV Client is using the Java programming language, version Java SE 8, and will adhere to the syntax and structure of java programming. The model will contain separate classes for the data types within the Bitcoin SPV Client. Classes and variables will be named

accordingly. For an example, the class pertaining to a Bitcoin Block will be labeled as "Block.java".

An example "Hello World" program can be seen below to demonstrate the syntax and standards of a class, method, variable and JavaDoc.

```
package BitcoinSPVClient.controller;

/** Program to display Hello World on the console.
 *
 * @author Frank Fasola
 * @author James Donnell
 * @author Spencer Escalante
 * @author Trevor Silva */
public class HelloWorld {

        /** A default "Hello World" String for displaying. */
        private static final String defaultHelloWorldString = "Hello World!";

        /** Displays default Hello World string in stdout.
         *
         * @param args Arguments not utilized. */
        public static void main (String[] args) {
                System.out.println(defaultHelloWorldString);
        }
}
```

6.3 Solution Data, Information View, and Data Requirements

Listed are selected classes which represent core parts of the project in which the other classes will be built around, interact with and rely on.

Class	Description	Variables	Variable Data Type
BlockChain.java	Represents all Blocks on the Blockchain	Blockchain	Map<Sha256Hash, Block>
Block.java	Represents a block in the blockchain, containing fields such as height, number of transactions and hash value.	Version BlockHash Merkle Root Time Difficulty Nonce Transactions	Long Sha256Hash Sha256Hash Long Long Long List<Transaction>

Wallet.java	Contains information for transactions with Bitcoin, such as amount of Bitcoin, owner and keys.	Transactions Description Version Unspent	Map<Sha256Hash, Transaction> String Integer Map<Sha256Hash, Transaction>
Transaction.java	Represents a single transaction.	Outputs OutputsTo Message	List<Long> List<Sha256Hash> String
Sha256Hash.java	Helper class for hashes as bytes.	Bytes	byte[]

Shown in **Appendix A** is a simple UML diagram to describe the basic relationship between the above classes. This will evolve over the course of the implementation phase. In the presented diagram, the Controller module would interact directly with the Blockchain and Wallet classes.

The GUI will include multiple components, mostly revolving around displaying the Blockchain and Wallet. Located within **Appendix B** is a mock-up visual of the current planned interface.

6.4 Module Description

The model aspect is the represented data received from full nodes on the blockchain. As information is requested by sending data to a full node, the client will also receive information. The data can represent many transactions inside a single block in the blockchain and transactions by a specified user, among other things. This data is the model for users to view activity on the blockchain.

The view component represents the graphical user interface which the user will see and interact with.The design of the GUI is not finalized and may change during the development of the project. The concept of the GUI is to have options for the user to select and tables to represent the data. The GUI will be implemented using the JavaFX and Swing libraries.

The client is the controller component of the MVC. The client will send and receive input and data, which will be represented in the model and shown to the user via the view component. The client can receive input from the user and send and receive data from nodes on the Bitcoin P2P network.

7 Implementation after 3 Weeks

Implementation of the Bitcoin client will be performed through pair programming, with the team split into groups of two, one person working on the GUI and the other on the backend for each respective section. Documentation for the project will be the responsibility of all team members.

The pairs and responsibilities are:

Section	Backend	GUI	Description
Blockchain	James	Trevor	The Blockchain section involves downloading the Bitcoin Blockchain headers through Peers and displaying their data in a consumer-friendly format.
Wallet	Frank	Spencer	The Wallet section involves utilizing the Blockchain and payment verification methods to display wallet transactions and balances in a format easily recognizable to users.

Week	James	Trevor	Frank	Spencer
Week 1	Connect to nodes and download block headers	Basic layout of Blockchain GUI	Basic wallet class implementation	Basic layout of Wallet GUI
Week 2	Ability to look up transactions through full nodes	Polished GUI layout and implementation	Wallet connects to client and network	Polished GUI layout and implementation
Week 3	Real time updating of blockchain in client	Connect blockchain and wallet GUI	Ability to send/receive Bitcoin	Connect blockchain and wallet GUI

8 Terms and Definitions

Term or Acronym	Definition
SPV	Simple Payment Verification
SDLC	Solution Development Lifecycle
P2P	Peer-to-Peer; usually in regards to a network topology
MVC	Model-View-Controller; a type of software design pattern
GUI	Graphical User Interface
OS	Operating System
Java SE 8	The current edition of the Java programming language; Java Standard Edition 8.
JavaDoc	Documentation of Java programming placed inline with related program code
UML	Unified Modeling Language

9 Supporting References

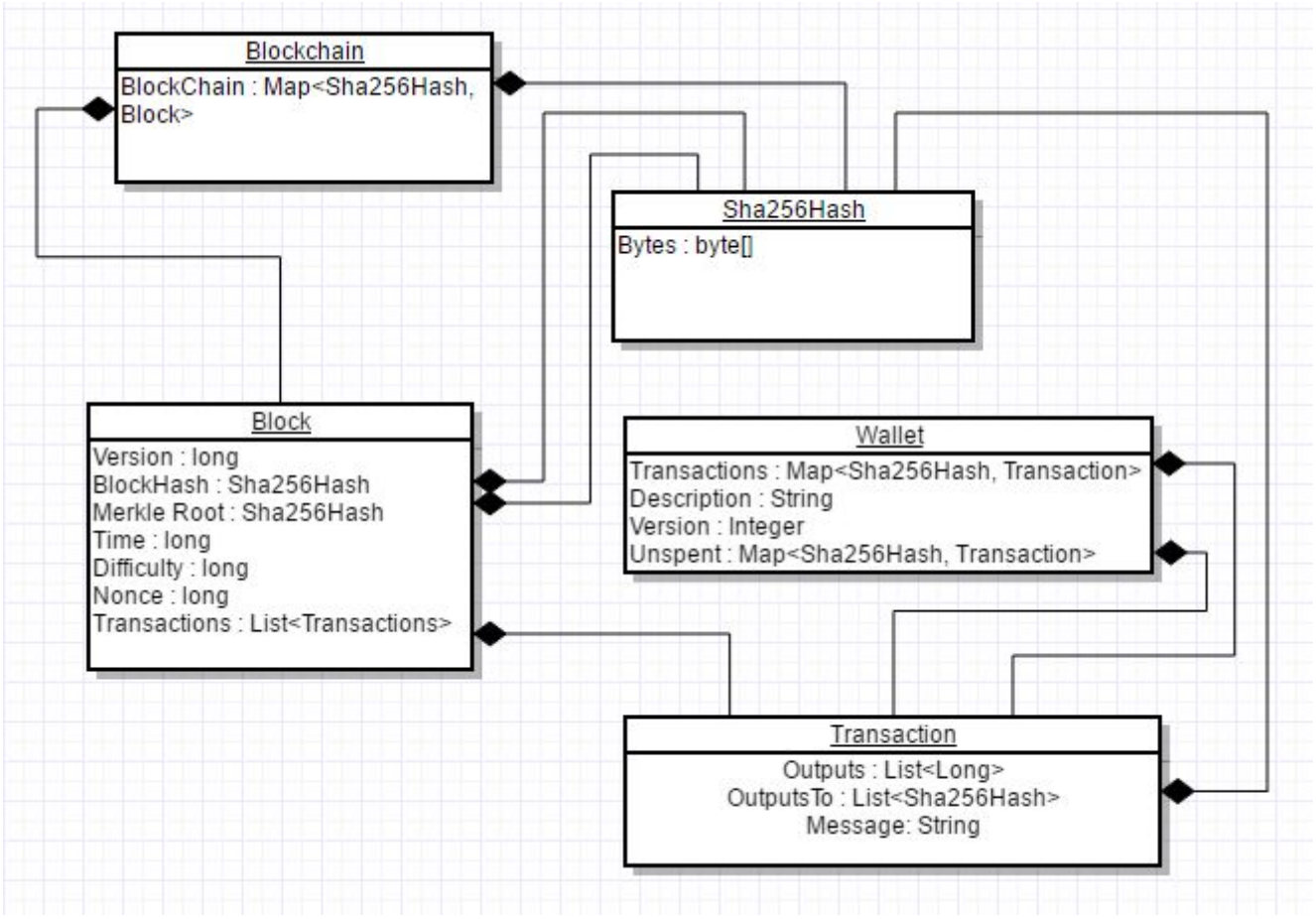
Reference	Description
https://bitcoin.org/en/	Website dedicated to Bitcoin with information on the basics to Bitcoin development.
https://www.bitcoin.com/	Website containing basic information and news about Bitcoin.
https://en.wikipedia.org/wiki/Bitcoin	Encyclopedia entry about Bitcoin.
http://www.coindesk.com/	Website containing news and price references for Bitcoin and digital currencies based off of Bitcoin.
https://bitcoinj.github.io/	Website for open-source project bitcoinj.

10 Revision History

Version	**Date**	**Revisions**
1.0	02/21/2017	Initial Release
1.1	03/01/2017	Updates after group discussion with Professor Bergmann

Appendix A

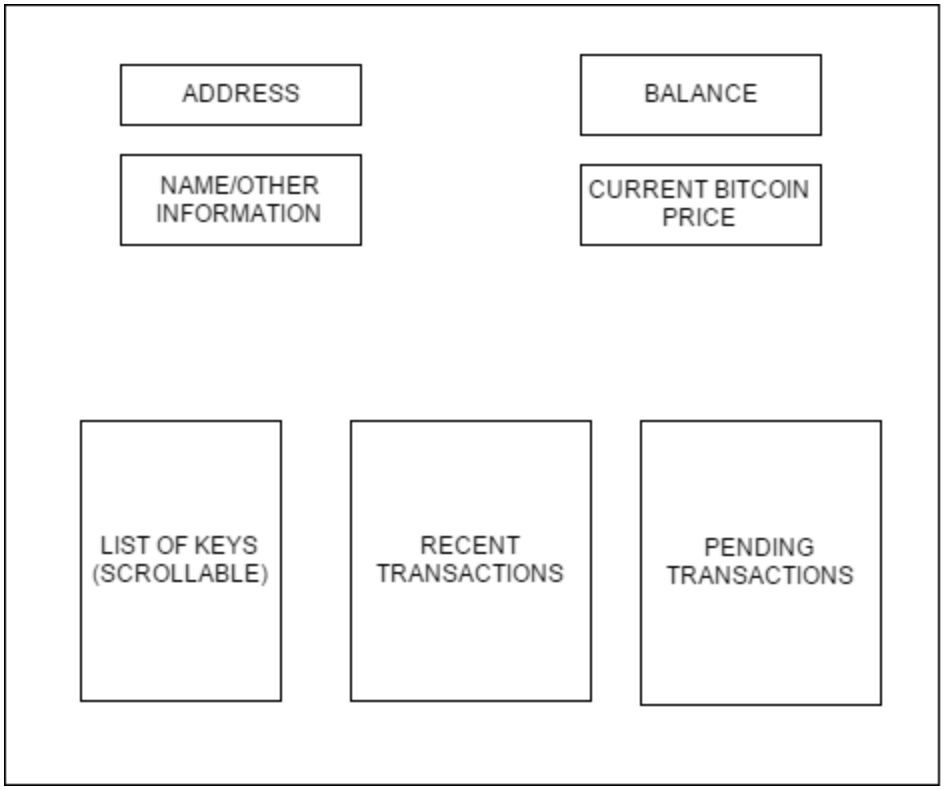
UML Diagram of core Java classes.



Appendix B

Visual mock-ups of GUI components,

Wallet GUI Concept



Blockchain GUI Concept