# Homework 6

- Honor Code: You must work completely independently on this assignment. Do not discuss the questions or answers with each other before the assignment is due. Any breach of the honor code will be handled per the University's policy on academic honesty.
- Follow the instructions very carefully. Answers that do not conform to the instructions will not be given credit.
- Submit your solutions through Blackboard as individual separate files, and not as zip archive.
- Understand thoroughly all the code given to you in this lab. Search for documentation online if there is a primitive or API you have not encountered before.
- Use Java 8. Only use the external Java libraries provided to you in the lab, if any.

1. Set the hw6.WalletInit.WALLETS_DIR constant to a directory that stores all the wallets.
2. Run the hw6.WalletInitTest class to ensure that you have enough testnet coins left from the previous homework. If you don't then, use the testnet faucet to get some.
3. Read and understand the ScriptTester abstract class, and the PayToPubKey subclass. The ScriptTester.run method creates two transactions to test the locking/unlocking scripts created in the PayToPubKey class. Give a detailed explanation on how ScriptTester.run works. Run the PayToPubKey subclass and give the two transaction IDs that it generates as your solution to question 3 in your hw6solutions.txt file. You must ensure that both transactions are confirmed by the network. You may need to run it a few times if they are not confirmed the first time. Use https://www.blocktrail.com/tBTC block chain explorer to find your transactions and check that your pay-to-public-key script appears in the transaction output of the first transaction.
   Note:
   - If you are having difficulty getting your transactions confirmed on testnet, first read the following article to understand why it's happening: https://www.dash.org/forum/threads/frozen-transactions-why-it-happens-and-how-to-fix-it.1649/.
   - One option is to submit the raw transaction bytes directly to a miner. You can use BitcoinJ's `Transaction.bitcoinSerialize` to generate a byte[] representation of your frozen transaction, convert it to hex using `DatatypeConverter.printHexBinary`, and submit the hex manually to bitcoin explorer for testnet (e.g. https://live.blockcypher.com/btc-testnet/pushtx/).
4. Implement the createLockingScript and createUnlockingScript methods in the PayToPubKeyHash class and run the class to test your pay-to-public-key-hash implementation. Give the two transaction IDs that it generates as your solution to question 4 in your hw6solutions.txt file. You must ensure that both transactions are confirmed by the network. No credit will be given to unconfirmed transactions. You may need to run it a few times if they are not confirmed the first time. Use https://www.blocktrail.com/tBTC block chain explorer to find your transactions and check that your pay-to-public-key-hash script appears in the transaction output of the first transaction.
5. Follow the same instructions in question 4, but for the MultiSigTransaction class. The locking script should lock the UTXO to a multisig script that is unlocked when exactly signature for one of the keys generated in the constructor (key1, key2, or key3) is provided in the unlocking script.
6. Follow the same instructions in question 4, but for the GroupMultiSigTransaction class. The locking script should lock the UTXO to a multi-sig transaction involving four parties such that the transaction can be redeemed by the first party (bank) combined with any one of the 3 others (customers) but not by only the customers or only the bank. Create and redeem the transaction and make sure that the script is as small as possible. You can use any legal combination of signatures to redeem the transaction but make sure that all combinations would have worked. The https://blocktrail.com/tBTC explorer won't show your script correctly. Instead try to use the https://test-insight.bitpay.com block explorer to make sure your scripts actually made it to the network.
7. Read and understand the hw6.SendMoneyBackToFaucet class. It sends your testnet coins back to the faucet. Run the class and give the transaction ID used to send your balance back to the faucet in your hw6solutions.txt file.