# Bitcoin and Cryptocurrency Technologies

# Assignment 3: Block Chain

In this assignment you will implement a node that's part of a block-chain-based distributed consensus protocol. Specifically, your code will receive incoming transactions and blocks and maintain an updated block chain.

**Files provided:**

| | |
|---|---|
| **Block.java** | Stores the block data structure. |
| **BlockHandler.java** | Uses BlockChain.java to process a newly received block, create a new block, or process a newly received transaction. |
| **ByteArrayWrapper.java** | A utility file which creates a wrapper for byte arrays such that it could be used as a key in hash functions. (See TransactionPool.java) |
| **Transaction.java** | This is similar to Transaction.java as provided in Assignment 1 except for introducing functionality to create a coinbase transaction. Take a look at Block.java constructor to see how a coinbase transaction is created. |
| **TransactionPool.java** | Implements a pool of transactions, required when creating a new block. |
| **UTXO.java** | From Assignment 1. |
| **UTXOPool.java** | From Assignment 1. |

**Files to be used from Assignment 1:**

**TxHandler.java:**

```
public class TxHandler {

    /* Creates a TxHandler whose current UTXO pool
       (collection of unspent transaction outputs) is utxoPool. */
    public TxHandler(UTXOPool utxoPool);

    /* Returns the current UTXO pool.
       If no outstanding UTXOs, returns an empty (non-null) UTXOPool object. */
    public UTXOPool getUTXOPool();

    /* Returns true iff
           (1) all outputs claimed by tx are in the current UTXO pool
           (2) the signatures on each input of tx are valid
           (3) no UTXO is claimed multiple times by tx, and
           (4) the sum of tx's input values >= sum of its output values */
    public boolean isValidTx(Transaction tx);

    /* Receives an unordered array possibleTxs of proposed transactions,
       checks each for correctness, returns a valid array of accepted transactions,
       and updates the current UTXO pool as appropriate. */
    public Transaction[] handleTxs(Transaction[] possibleTxs);
}
```

Create a public function getUTXOPool() in the TxHandler.java you have created for assignment 1 and copy the file to your code for assignment 3.

We will run your code with our reference TxHandler.java. We are providing you with a .jar file containing compiled version of reference TxHandler.java along with Transaction.java, UTXO.java and UTXOPool.java. You can use this for testing too.

**File to be modified:**

**BlockChain.java**

```
/* BlockChain should maintain enough blocks in memory to implement the
   functions below, but should not keep the entire block chain in memory */

public class BlockChain {
    public static final int CUT_OFF_AGE = 10;

    // Code provided to you
    private class BlockNode {}

    /* Create an empty block chain with just a genesis block.
     * Assume genesis block is a valid block containing nothing but a
     * coinbase transaction.
     */
    public BlockChain(Block genesisBlock) {
       // IMPLEMENT THIS
    }

    /* Get the maximum height block
     */
    public Block getMaxHeightBlock() {
       // IMPLEMENT THIS
    }

    /* Get the UTXOPool for mining a new block on top of
     * max height block
     */
    public UTXOPool getMaxHeightUTXOPool() {
       // IMPLEMENT THIS
    }
    /* Get the transaction pool to mine a new block
     */
    public TransactionPool getTransactionPool() {
       // IMPLEMENT THIS
    }

    /* Add a block to block chain if it is valid.
     * For validity, all transactions should be valid
     * and block should be at height > (maxHeight - CUT_OFF_AGE).
     * For example, you can try creating a new block over genesis block
     * (block height 2) if blockChain height is <= CUT_OFF_AGE + 1.
     * As soon as height > CUT_OFF_AGE + 1,you cannot create a new block at height 2.
     * Return true of block is successfully added
     */
    public boolean addBlock(Block b) {
       // IMPLEMENT THIS
    }

    /* Add a transaction in transaction pool
     */
    public void addTransaction(Transaction tx) {
       // IMPLEMENT THIS
    }
}
```

The BlockChain class is responsible for maintaining a block chain. Since the entire block chain could be huge in size, you should only keep around the most recent blocks. The exact number to store is up to your design, as long as you're able to implement all the API functions.

Since there can be (multiple) forks, blocks form a tree rather than a list. Your design should take this into account. You have to maintain a UTXO pool corresponding to every block on top of which a new block might be created.

**Assumptions and Guidelines:**

- A new genesis block would not be mined. If you receive a block which claims to be a genesis block (parent is a null hash) in the `addBlock(Block b)` function, you can return false.
- If there are multiple blocks at the same height, return the oldest block in `getMaxHeightBlock()` function.
- Assume for simplicity, a coinbase transaction of a block is available to be used in the next block mined on top of it. (This is contrary to the actual Bitcoin protocol when there is a gap of 100 blocks only after which the coinbase transaction can be used).
- Maintain only one global Transaction Pool for the block chain and keep adding transactions to it on receiving transactions and keep removing transactions from it if a new block is received or created. This might cause some transactions to be lost. For example, a block is received on Chain A including transaction Tx1. We remove Tx1 from the transaction pool. Now suppose chain B offshoots chain A. Do not put Tx1 back in the pool, although ideally it should be put back in. This is to simplify your work as well as our work in testing. (Miners are not responsible for including transactions in the blocks. If a transaction is lost, it is the responsibility of the transaction owner to re-broadcast it in the network).
- The coinbase value is kept as constant in our entire block chain (= 25 bitcoins) whereas we know it changes every four years.
- When checking for validity of a newly received block, just checking if the transactions form a valid set is enough. The set need not be a maximum possible set of transactions. Also, you should not check for hash of the block to contain specific zeros (no proof of work here).