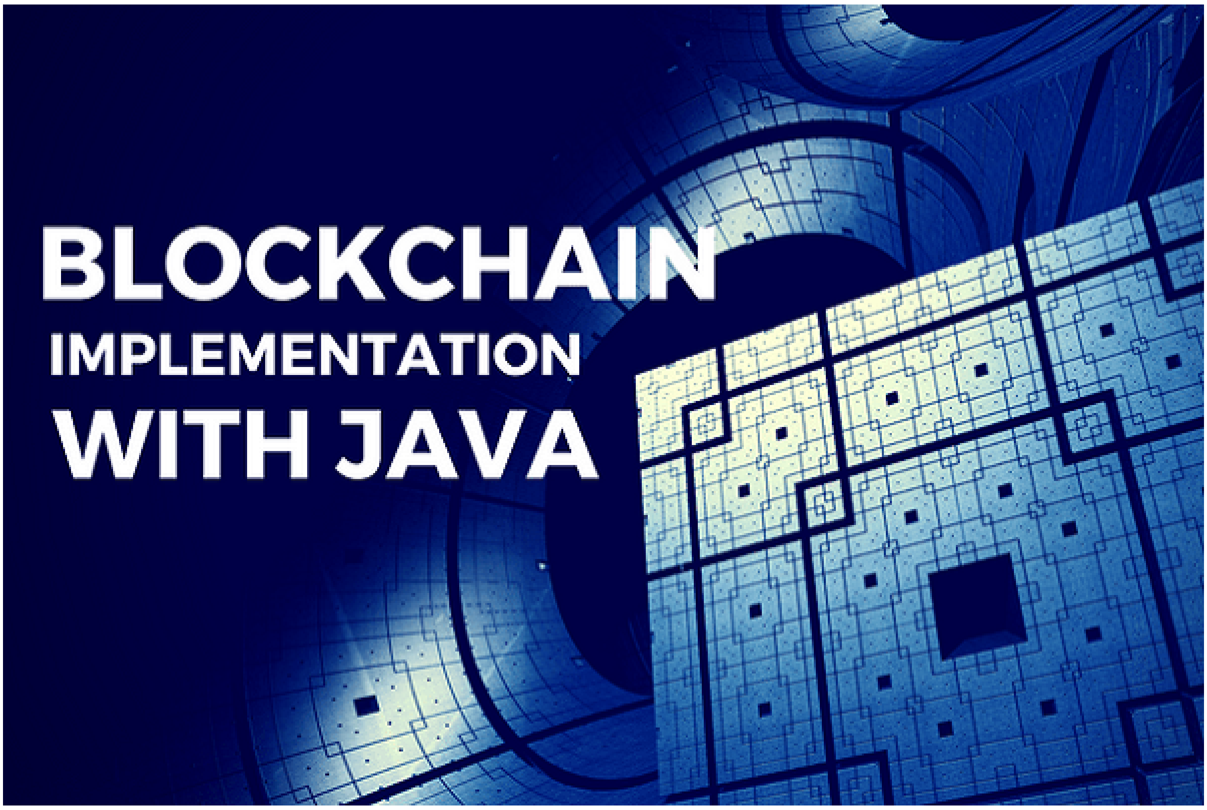
[illegible]

[illegible]


BLOCKCHAIN
IMPLEMENTATION
WITH JAVA

[illegible]

[illegible] **213** [illegible]

[illegible]

[illegible]

[illegible]

[illegible] SHA-256 [illegible]

```
...
public class Block&lt;T extends Tx&gt; {
public long timeStamp;
private int index;
private List&lt;T&gt; transactions = new ArrayList&lt;T&gt;();
private String hash;
private String previousHash;
private String merkleRoot;
private String nonce = “0000”;
// caches Transaction SHA256 hashes
    public Map&lt;String,T&gt; map = new HashMap&lt;String,T&gt;();
...
```

[illegible] Tx [illegible] PreviousHash [illegible]

Block Hash

[illegible] Block Hash [illegible] SHA-256 [illegible]
[illegible] Block.java [illegible]

```
...
public void computeHash() {
     Gson parser = new Gson(); // probably should cache this instance
     String serializedData = parser.toJson(transactions);
     setHash(SHA256.generateHash(timeStamp + index + merkleRoot + serializedData + nonce +
previousHash));
     }
...
```

[illegible] JSON [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible] SimpleBlockChain.java [illegible]:

```
...
...
public class SimpleBlockchain<T extends Tx> {
public static final int BLOCK_SIZE = 10;
public List<Block<T>> chain = new ArrayList<Block<T>>();
public SimpleBlockchain() {
// create genesis block
chain.add(newBlock());
}
...
```

[illegible] Chain [illegible] Tx [illegible] No arg [illegible] newBlock() [illegible]:

```
...
public Block<T> newBlock() {
int count = chain.size();
String previousHash = “root”;
if (count > 0)
previousHash = blockChainHash();
Block<T> block = new Block<T>();
block.setTimeStamp(System.currentTimeMillis());
block.setIndex(count);
block.setPreviousHash(previousHash);
return block;
}
...
```

[illegible].

[illegible] (head) [illegible] SimpleBlockchain.java [illegible]:

```
....
public void addAndValidateBlock(Block<T> block) {
// compare previous block hash, add if valid
Block<T> current = block;
for (int i = chain.size() - 1; i >= 0; i--) {
Block<T> b = chain.get(i);
if (b.getHash().equals(current.getPreviousHash())) {
current = b;
} else {
```

```
throw new RuntimeException(“Block Invalid”);
}
}
this.chain.add(block);
}
...
```

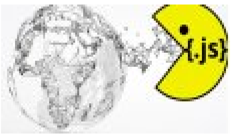
[illegible].

[illegible] validate SimpleBlockChain.java() [illegible]:

```
...
public boolean validate() {
String previousHash = null;
for (Block<T> block : chain) {
String currentHash = block.getHash();
if (!currentHash.equals(previousHash)) {
return false;
}
previousHash = currentHash;
}
return true;
}
...
```

[illegible].

[illegible]



[illegible]

[illegible] ([illegible]) [illegible]

[illegible]

[illegible] API [illegible] HTTP [illegible].

[illegible] Merkle Tree

[illegible] Merkle Tree [illegible] Merkle Root [illegible] Merkle [illegible].

[illegible]. [illegible]
[illegible] Merkle Root [illegible].

[illegible] Block.java [illegible] Merkle Tree [illegible]:

```
...
public List<String> merkleTree() {
ArrayList<String> tree = new ArrayList<>();
// Start by adding all the hashes of the transactions as leaves of the
// tree.
for (T t : transactions) {
tree.add(t.hash());
}
int levelOffset = 0; // Offset in the list where the currently processed
// level starts.
// Step through each level, stopping when we reach the root (levelSize
// == 1).
for (int levelSize = transactions.size(); levelSize > 1; levelSize = (levelSize + 1) / 2) {
// For each pair of nodes on that level:
for (int left = 0; left < levelSize; left += 2) {
// The right hand node can be the same as the left hand, in the
// case where we don’t have enough
// transactions.
int right = Math.min(left + 1, levelSize - 1);
String tleft = tree.get(levelOffset + left);
String tright = tree.get(levelOffset + right);
tree.add(SHA256.generateHash(tleft + tright));
}
// Move to the next level.
levelOffset += levelSize;
}
return tree;
}
...
```

[illegible] Merkle Tree [illegible]. [illegible] Merkle Tree [illegible] Merkle Roots [illegible]. [illegible] Merkle Tree [illegible].

```
...
Test@
public void merkleTreeTest() {
// create chain, add transaction
SimpleBlockchain<Transaction> chain1 = new SimpleBlockchain<Transaction>();
chain1.add(new Transaction(“A”)).add(new Transaction(“B”)).add(new Transaction(“C”)).add(new
Transaction(“D”));
// get a block in chain
Block<Transaction> block = chain1.getHead();
System.out.println(“Merkle Hash tree :” + block.merkleTree());
// get a transaction from block
Transaction tx = block.getTransactions().get(0);
```

```
// see if block transactions are valid, they should be
block.transasctionsValid();
assertTrue(block.transasctionsValid());
// mutate the data of a transaction
tx.setValue("Z");
// block should no longer be valid, blocks MerkleRoot does not match computed merkle tree of
transactions
;(()assertFalse(block.transasctionsValid
{
...
```

[illegible] (Merkle Root [illegible]) [illegible]

[illegible]

[illegible]

[illegible]

mining

[illegible] Mining [illegible].

[illegible] Mining [illegible] Miner.java [illegible] (proofOfWork(Block block [illegible]:

```
private String proofOfWork(Block block) {
String nonceKey = block.getNonce();
long nonce = 0;
boolean nonceFound = false;
String nonceHash = "";
Gson parser = new Gson();
String serializedData = parser.toJson(transactionPool);
String message = block.getTimeStamp() + block.getIndex() + block.getMerkleRoot() +
serializedData
+ block.getPreviousHash();
while (!nonceFound) {
nonceHash = SHA256.generateHash(message + nonce);
nonceFound = nonceHash.substring(0, nonceKey.length()).equals(nonceKey);
nonce++;
}
return nonceHash;
}
```

[illegible] SHA-256 [illegible]. [illegible]. [illegible]. [illegible].

[illegible]:
[illegible]
[illegible]:
[illegible]
[illegible]:
06:50 - 16/02/1398
[illegible]:
[illegible] - [illegible] - [illegible]

[illegible]
[illegible]: https://www.shabakeh-mag.com/cover-story/14856/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87%E2%80%8C%D8%B3%D8%A7%D8%B2%DB%8C-%D8%B2%D9%86%D8%AC%D9%8A%D8%B1%D9%87-%D8%A8%D9%84%D9%88%DA%A9%DB%8C-%D8%A8%D8%A7-%DA%A9%D8%AF%D9%87%D8%A7%DB%8C-%D8%AC%D8%A7%D9%88%D8%A7