```java
package noobchain;

import java.util.ArrayList;
import java.util.Date;

public class Block {

	// reviewed
	public String hash;  // hash for this block (previousHash data timeStamp)
	public String previousHash; // has for previous block
	private String data;
	private long timeStamp;
	private int nonce;

	public ArrayList<Transaction> transactions=new ArrayList<Transaction>();


	public Block(String previousHash) {
		this.previousHash=previousHash;
		this.timeStamp=new Date().getTime();
		this.hash=calculateHash();
	}

	// reviewed
	public Block(String data, String perviousHash) {
		//this.data=data;
		this.previousHash=perviousHash; // hash of previous block
		this.timeStamp=new Date().getTime();
		this.hash=calculateHash();
	}

	public String calculateHash() {
		String calculatehash=StringUtil.applySha256(  // create hash from
				previousHash+                                    // previousHash, timeStamp
				Long.toString(timeStamp)+                    // data
				Integer.toString(nonce)+
				data);

		return calculatehash;

	}

	public void mineBlock(int difficult) {
		String target=new String(new char[difficult]).replace('\0', '0');  // create leading zeros
		System.out.println("\n"+target);
		while(!hash.substring(0, difficult).equals(target)) { // if hash match found
				nonce++;
				hash=calculateHash();  // if not, keep mining.
		}
		System.out.println("Block mined!! : "+hash);
	}

	public boolean addTransaction(Transaction transaction) {
		if(transaction==null) return false;
		if(previousHash!="0") {
			if(transaction.processTransaction()!=true) {
				System.out.println("Transaction failed to process.  Discarded");
				return false;
			}
```

```
                }

                transactions.add(transaction);
                System.out.println("Transaction successfully added to the Block");
                return true;
        }
}
```

```java
package noobchain;

import java.security.Key;
import java.security.MessageDigest;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.ArrayList;
import java.util.Base64;


public class StringUtil {

	// reviewed
	public static String applySha256(String input) {
		try {

			MessageDigest digest=MessageDigest.getInstance("SHA-256");
			byte[] hash=digest.digest(input.getBytes("UTF-8"));
			StringBuffer hexString=new StringBuffer();

			for(int i=0; i< hash.length; i++) {
				String hex=Integer.toHexString(0xFF & hash[i]);  // remove sign extension | leading zero
				if(hex.length() == 1) hexString.append('0');
				hexString.append(hex);
			}
			return hexString.toString();
		}
		catch(Exception e) {
			throw new RuntimeException(e);
		}
	}

	// reviewed
	public static byte[] applyEDSASig(PrivateKey privateKey, String input) {
		Signature dsa;
		byte[] output=new byte[0];

		try {
			dsa=Signature.getInstance("ECDSA", "BC");
			dsa.initSign(privateKey);
			byte[] strByte=input.getBytes();
			dsa.update(strByte);
			byte[] realSig=dsa.sign();
			output=realSig;
		} catch(Exception e) {
			throw new RuntimeException(e);
		}
		return output;
	}

	public static boolean verifyECDSASig(PublicKey publicKey, String data, byte[] signature) {
		try {
			Signature ecdsaVerify=Signature.getInstance("ECDSA", "BC");
			ecdsaVerify.initVerify(publicKey);
			ecdsaVerify.update(data.getBytes());
			return ecdsaVerify.verify(signature);
		} catch(Exception e) {
			throw new RuntimeException(e);
```

```
            }
        }

        public static String getStringFromKey(Key key) {
            return Base64.getEncoder().encodeToString(key.getEncoded());
        }

        // reviewed
        public static String getMerkleRoot(ArrayList<Transaction> transactions) {
            int count=transactions.size();
            ArrayList<String> previousTreeLayer=new ArrayList<String>();
            for(Transaction transaction: transactions) {
                previousTreeLayer.add(transaction.transactionId);
            }

            ArrayList<String> treeLayer=previousTreeLayer;
            while(count>1) {
                treeLayer=new ArrayList<String>();
                for(int i=1; i < previousTreeLayer.size(); i++) {
                    treeLayer.add(applySha256(previousTreeLayer.get(i-1)+
                                    previousTreeLayer.get(i)));
                }
                count=treeLayer.size();
                previousTreeLayer=treeLayer;
            }
            String merkleRoot=(treeLayer.size()==1?treeLayer.get(0):"");
            return merkleRoot;
        }
}
```

```
package noobchain;

import java.security.Security;
import java.util.ArrayList;
import java.util.HashMap;

import com.google.gson.GsonBuilder;



public class NoobChain {

	public static ArrayList<Block> blockchain=new ArrayList<Block>();
	public static HashMap<String, TransactionOutput> UTXOs= new HashMap<String, TransactionOutput>();
	public static float minimumTransaction=0.1f;

	public static int difficulty=3;
	public static Wallet walletA;
	public static Wallet walletB;

	public static Transaction genesisTransaction;

	public static void main(String [] args) {
//		Block genesisBlock=new Block("Hi I am the first block", "0");
//		System.out.println("Hash for block 1: "+genesisBlock.hash);
//
//		Block secondBlock=new Block("Yo I am the second block", genesisBlock.hash);
//		System.out.println("Hash for block 1: "+secondBlock.hash);
//
//		Block thirdBlock=new Block("Hey I am the third block", secondBlock.hash);
//		System.out.println("Hash for block 1: "+thirdBlock.hash);
```

```
//              blockchain.add(new Block("Hi im the first block", "0"));
//              System.out.print("Tryuing to Mine block 1...");
//              blockchain.get(0).mineBlock(difficulty);
//
//              blockchain.add(new Block("Yo im the second block", blockchain.get(blockchain.size()-1).hash));
//              System.out.print("Tryuing to Mine block 2...");
//              blockchain.get(1).mineBlock(difficulty);
//
//              blockchain.add(new Block("Hey im the second block", blockchain.get(blockchain.size()-1).hash));
//              System.out.print("Tryuing to Mine block 3...");
//              blockchain.get(2).mineBlock(difficulty);
//
//              System.out.println("\nBlockchain is valid: "+isChainValid());
//
//              String blockchainJson=new GsonBuilder().setPrettyPrinting().create().toJson(blockchain);
//              System.out.println("\nThe block chain");
//              System.out.println(blockchainJson);

                // Test wallet and transactions

//              Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
//
//              walletA=new Wallet();
//              walletB=new Wallet();
//
//              System.out.println("Private and public keys: ");
//              System.out.println(StringUtil.getStringFromKey(walletA.privateKey));
//              System.out.println(StringUtil.getStringFromKey(walletA.publicKey));
//
```

```
//              Transaction transaction=new Transaction(walletA.publicKey, walletB.publicKey, 5, null);
//              transaction.generateSignature(walletA.privateKey);
//              transaction.value=50;
//              System.out.println("Is signature verified");
//              System.out.println(transaction.verifySignature());

                // Finale

                Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());

                walletA=new Wallet();
                walletB=new Wallet();
                Wallet coinbase=new Wallet();



                genesisTransaction=new Transaction(coinbase.publicKey, walletA.publicKey, 100f, null);
                genesisTransaction.generateSignature(coinbase.privateKey);
                genesisTransaction.transactionId="0";
                genesisTransaction.outputs.add(new TransactionOutput(genesisTransaction.recipient,
                                genesisTransaction.value, genesisTransaction.transactionId));
                UTXOs.put(genesisTransaction.outputs.get(0).id, genesisTransaction.outputs.get(0));
                System.out.println("Creating and Mining Genesis block...");
                Block genesis=new Block("0");

                Block block1=new Block(genesis.hash);

                System.out.println("\nWalletA's balance is: "+walletA.getBalance());
                System.out.println("\nWalletA is attempting to send funds (40); to WalletB...");

                block1.addTransaction(walletA.sendFunds(walletB.publicKey, 40f));
```

```
		addBlock(block1);
		System.out.println("\nWalletA's balance is: "+walletA.getBalance());
		System.out.println("\nWalletB's balance is: "+walletB.getBalance());


		Block block2=new Block(block1.hash);

		System.out.println("\nWalletA is attempting to send more funds (1000)then it has...");

		block2.addTransaction(walletA.sendFunds(walletB.publicKey, 1000f));
		addBlock(block2);
		System.out.println("\nWalletA's balance is: "+walletA.getBalance());
		System.out.println("\nWalletB's balance is: "+walletB.getBalance());


		Block block3=new Block(block2.hash);

		System.out.println("\nWalletB is attempting to send more funds (20) to WalletA...");

		block3.addTransaction(walletB.sendFunds(walletA.publicKey, 20f));
		addBlock(block3);
		System.out.println("\nWalletA's balance is: "+walletA.getBalance());
		System.out.println("\nWalletB's balance is: "+walletB.getBalance());

		isChainValid();
	}

	// reviewed - Valid / but not verify
	public static Boolean isChainValid() {
//		Block currentBlock;
```

```
//             Block previousBlock;
//             String hashTarget = new String(new char[difficulty]).replace('\0', '0');
//
//             for(int i=1; i <blockchain.size(); i++) {
//                     currentBlock=blockchain.get(i);
//                     previousBlock=blockchain.get(i-1);
//
//                     if(!previousBlock.hash.equals(currentBlock.previousHash)) {
//                             System.out.println("Previous hashes not equal");
//                             return false;
//                     }
//
//                     if(!currentBlock.hash.substring(0, difficulty).equals(hashTarget)) {
//                             System.out.println("This block hasn't been mined");
//                             return false;
//                     }
//             }

//             return true;


             // final version

             Block currentBlock;
             Block previousBlock;

             String hashTarget=new String(new char[difficulty]).replace('\0',  '0');
             HashMap<String, TransactionOutput> tempUTXOs=new HashMap<String, TransactionOutput>();
             tempUTXOs.put(genesisTransaction.outputs.get(0).id, genesisTransaction.outputs.get(0));
```

```
for(int i=1; i < blockchain.size(); i++) {
        currentBlock=blockchain.get(i);
        previousBlock=blockchain.get(i-1);

        if(!currentBlock.hash.equals(currentBlock.calculateHash())) {
                System.out.println("Current Hashes not equal");
                return false;
        }

        if(!previousBlock.hash.equals(currentBlock.previousHash)) {
                System.out.println("Previous Hashes not equal");
                return false;
        }

        if(!currentBlock.hash.substring(0, difficulty).equals(hashTarget)) {
                System.out.println("This block hasn't been mined.");
                return false;
        }

        TransactionOutput tempOutput;
        for(int t=0; t<currentBlock.transactions.size(); t++) {
                Transaction currentTransaction=currentBlock.transactions.get(t);

                if(!currentTransaction.verifySignature()) {
                        System.out.println("Signature of Transaction ("+t+" ) is invalid.");
                }

                if(currentTransaction.getInputsValue()!= currentTransaction.getOutputsValue()) {
                        System.out.println("Input are not equal to outputs on transaction.");
                }
```

```
		for(TransactionInput input: currentTransaction.inputs) {
			tempOutput=tempUTXOs.get(input.transactionOutputId);

			if(tempOutput == null) {
				System.out.println("Referenced input on Transaction ( "+t+" ) is missing.");
				return false;
			}

			if(input.UTXO.value != tempOutput.value) {
				System.out.println("Referenced input on Transaction ( "+t+" ) value is invalid.");
			}

			tempUTXOs.remove(input.transactionOutputId);
		}

		for(TransactionOutput output: currentTransaction.outputs) {
			tempUTXOs.put(output.id, output);
		}

		if(currentTransaction.outputs.get(0).recipient!= currentTransaction.recipient) {
			System.out.println("Transaction ( "+t+" ) output recipient is not who it should be");
			return false;
		}

		if(currentTransaction.outputs.get(1).recipient!= currentTransaction.sender) {
			System.out.println("Transaction ( "+t+" ) output 'change' is not sender");
			return false;
		}
	}
```

```
                }


                System.out.println("Blockchain is valid.");
                return true;
        }

        public static void addBlock(Block newBlock) {
                newBlock.mineBlock(difficulty);
                blockchain.add(newBlock);
        }
}
```

```java
package noobchain;
import java.security.*;
import java.security.spec.ECGenParameterSpec;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class Wallet {

	public PrivateKey privateKey;
	public PublicKey publicKey;

	public Wallet() {
		generateKeyPair();
	}

	public HashMap<String,TransactionOutput> UTXOs =
			new HashMap<String,TransactionOutput>();

	// reviewed
	public void generateKeyPair() {
		try {
			KeyPairGenerator keyGen=KeyPairGenerator.getInstance("ECDSA", "BC");
			SecureRandom random=SecureRandom.getInstance("SHA1PRNG");
			ECGenParameterSpec ecSpec=new ECGenParameterSpec("prime192v1");

			keyGen.initialize(ecSpec, random);
			KeyPair keyPair=keyGen.generateKeyPair();
			privateKey=keyPair.getPrivate();
			publicKey=keyPair.getPublic();
		}
		catch(Exception e) {
			throw new RuntimeException(e);
		}
	}

	// reviewed
	public float getBalance() {
		float total=0;
		for(Map.Entry<String, TransactionOutput> item:NoobChain.UTXOs.entrySet()){
			TransactionOutput UTXO=item.getValue();
			if(UTXO.isMine(publicKey)) {
				UTXOs.put(UTXO.id, UTXO);
				total+=UTXO.value;
			}
		}
		return total;
	}

	// reviewed
	public Transaction sendFunds(PublicKey _receipient, float value) {
		if(getBalance() < value) {
			System.out.println("Not enough funds to send transaction. Transaction discarded.");
			return null;
		}

		ArrayList<TransactionInput> inputs=new ArrayList<TransactionInput>();

		float total=0;
```

```
		for(Map.Entry<String, TransactionOutput> item: UTXOs.entrySet()){
			TransactionOutput UTXO=item.getValue();
			total+=UTXO.value;
			inputs.add(new TransactionInput(UTXO.id));
			if(total>value) break;
		}

		Transaction newTransaction=new Transaction(publicKey, _receipient, value, inputs);
		newTransaction.generateSignature(privateKey);

		for(TransactionInput input: inputs) {
				UTXOs.remove(input.transactionOutputId);
		}

		return newTransaction;
	}
}
```

```java
package noobchain;

import java.security.*;
import java.util.ArrayList;

public class Transaction {

	public String transactionId;
	public PublicKey sender;
	public PublicKey recipient;
	public float value;
	public byte[] signature;


	public ArrayList<TransactionInput> inputs=new ArrayList<TransactionInput>();
	public ArrayList<TransactionOutput> outputs=new ArrayList<TransactionOutput>();

	private static int sequence=0;

	// reviewed
	public Transaction(PublicKey from, PublicKey to, float value, ArrayList<TransactionInput> inputs) {
		this.sender=from;
		this.recipient=to;
		this.value=value;
		this.inputs=inputs;
	}

	// reviewed
	private String calculateHash() {
		sequence++;
		return StringUtil.applySha256(
				StringUtil.getStringFromKey(sender)+
				StringUtil.getStringFromKey(recipient)+
				Float.toString((value)+sequence
				));
	}

	public void generateSignature(PrivateKey privateKey) {
		String data=StringUtil.getStringFromKey(sender)+
				StringUtil.getStringFromKey(recipient)+
				Float.toString(value);
		signature=StringUtil.applyEDSASig(privateKey, data);
	}

	public boolean verifySignature() {
		String data=StringUtil.getStringFromKey(sender)+
				StringUtil.getStringFromKey(recipient)+
				Float.toString(value);
		return StringUtil.verifyECDSASig(sender,  data,  signature);
	}

	// reviewed
	public boolean processTransaction() {
		if(verifySignature()==false) {
			System.out.println("#Tranaction Signature failed to verify");
			return false;
		}

		for(TransactionInput i : inputs) {
```

```
                i.UTXO=NoobChain.UTXOs.get(i.transactionOutputId);
            }

            if(getInputsValue() < NoobChain.minimumTransaction) {
                System.out.println("#Transaction inputs to small: "+getInputsValue());
                return false;
            }

            float leftOver=getInputsValue()-value;
            transactionId=calculateHash();
            outputs.add(new TransactionOutput(this.recipient, value, transactionId));
            outputs.add(new TransactionOutput(this.sender, leftOver, transactionId));

            for(TransactionOutput o: outputs) {
                NoobChain.UTXOs.put(o.id, o);
            }

            for(TransactionInput i:inputs) {
                if(i.UTXO==null) continue;
                NoobChain.UTXOs.remove(i.UTXO.id);
            }
            return true;
        }

        public float getInputsValue(  ) {
            float total=0;

            for(TransactionInput i: inputs) {
                if(i.UTXO==null) continue;
                total+=i.UTXO.value;

            }
            return total;
        }

        public float getOutputsValue() {
            float total=0;
            for(TransactionOutput o: outputs) {
                total+=o.value;
            }
            return total;
        }
}
```

```java
package noobchain;

public class TransactionInput {
	public String transactionOutputId;
	public TransactionOutput UTXO;

	public TransactionInput(String transactionOutputId) {
		this.transactionOutputId=transactionOutputId;
	}

}
```

```java
package noobchain;

import java.security.PublicKey;

public class TransactionOutput {

	public String id;
	public PublicKey recipient;
	public Float value;
	public String parentTransactionId;

	public TransactionOutput(PublicKey recipient, float value, String parentTransactionId) {
		this.recipient=recipient;
		this.value=value;
		this.parentTransactionId=parentTransactionId;
		this.id=StringUtil.applySha256(StringUtil.getStringFromKey(recipient)+Float.toString(value)+
				parentTransactionId);
	}

	public boolean isMine(PublicKey publicKey) {
		return (publicKey==recipient);
	}
}
```