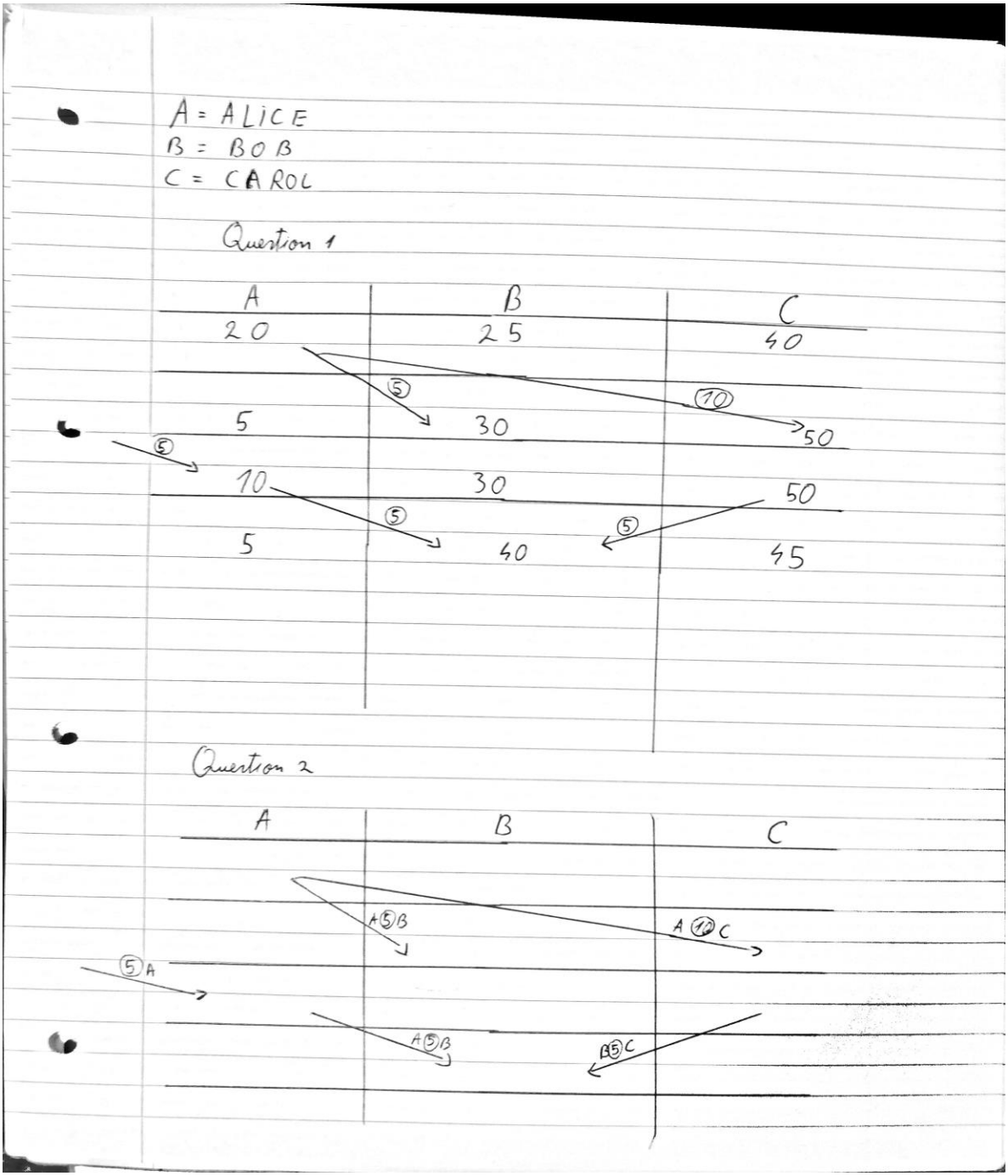
CW1 – CSCM29

Lab 1.

Question 1. and Question 2. (see scanned image and photo below)



A = ALICE
B = BOB
C = CAROL

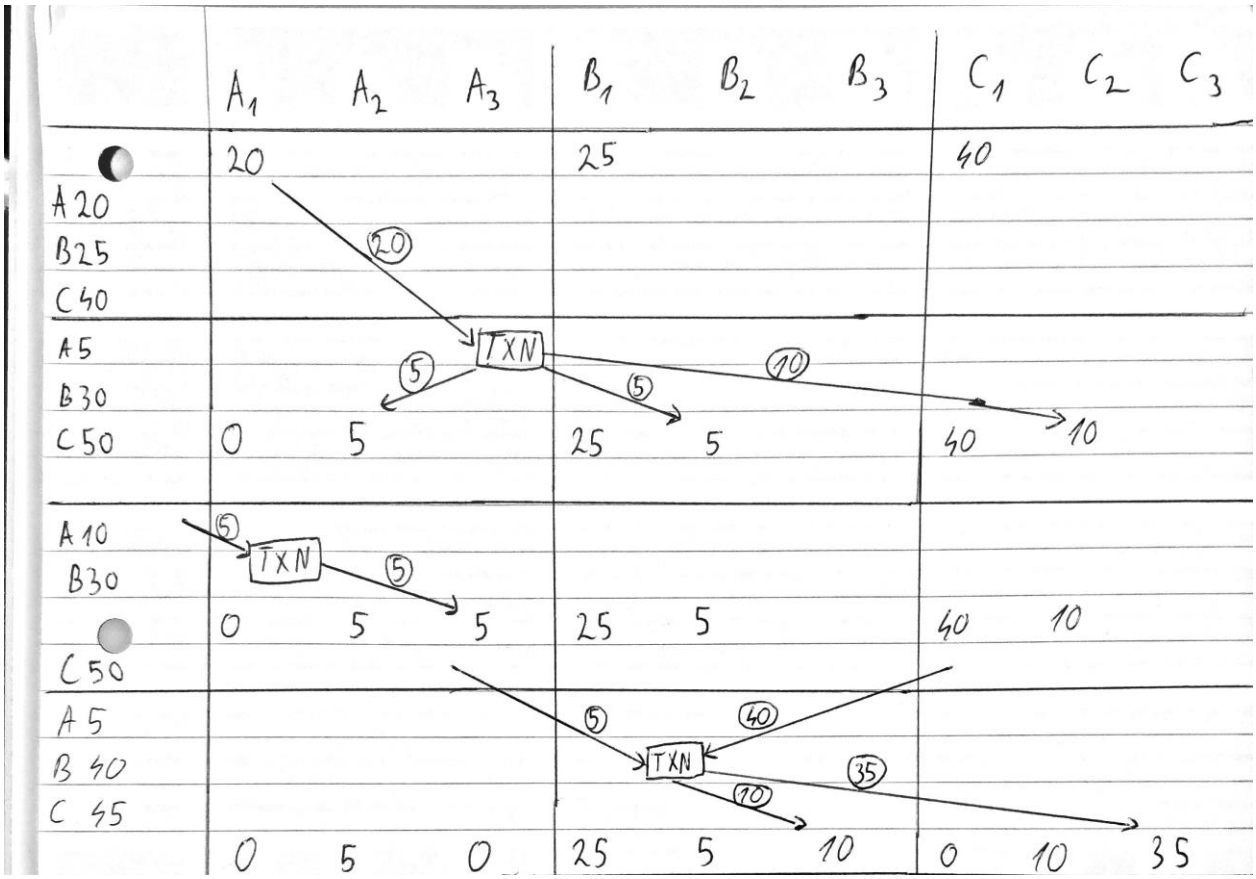
Question 1

A	B	C
20	25	40
5	30	50
10	30	50
5	40	45

Question 2

A	B	C
	A 5 B	A 10 C
5 A		
	A 5 B	B 5 C

Question 3. (see scanned image and photo below)



	A_1	A_2	A_3	B_1	B_2	B_3	C_1	C_2	C_3
	20			25			40		
A 20									
B 25									
C 40									
A 5									
B 30									
C 50	0	5		25	5		40	10	
A 10									
B 30									
	0	5	5	25	5		40	10	
C 50									
A 5									
B 40									
C 45									
	0	5	0	25	5	10	0	10	35

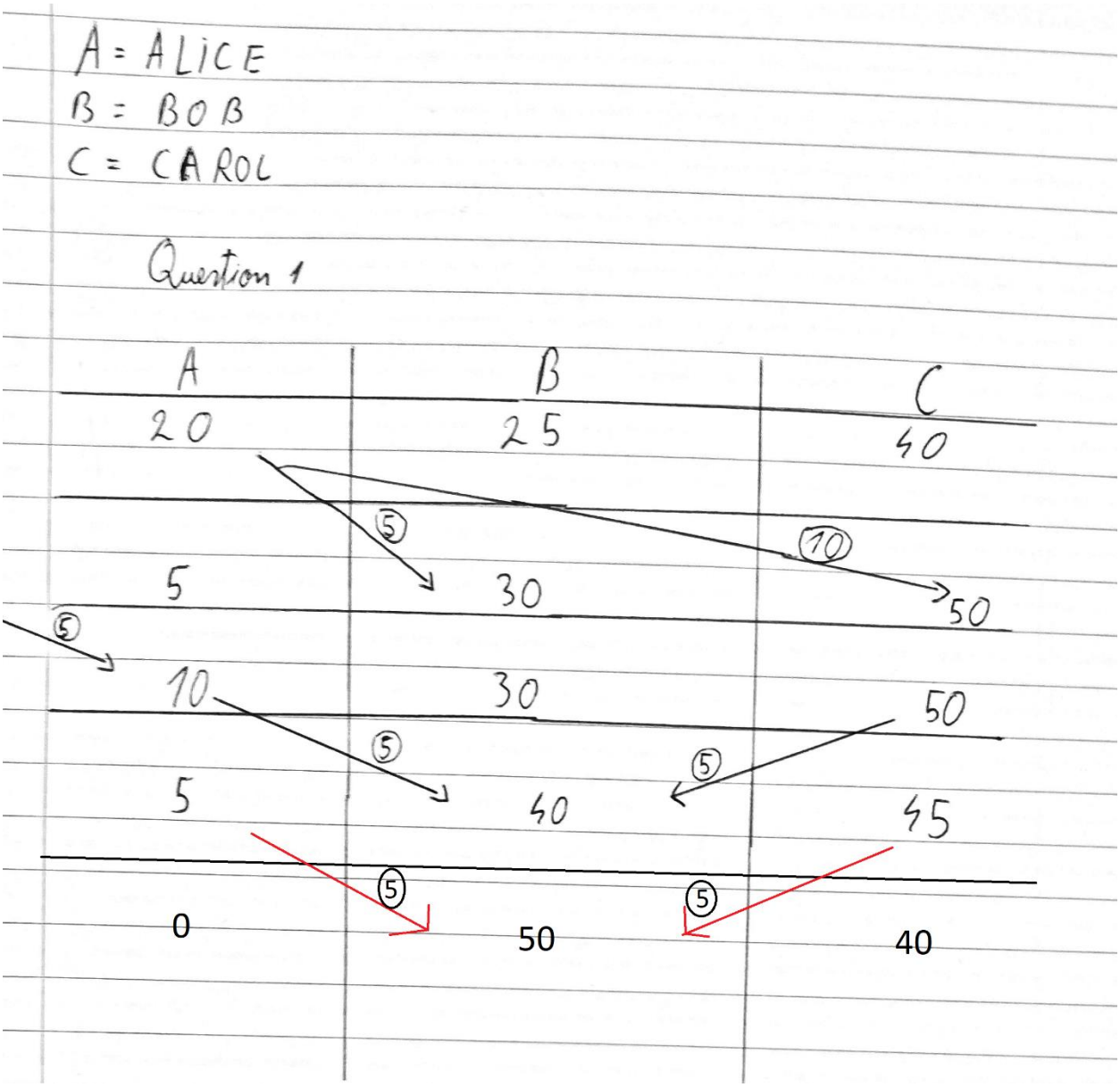
TXN (20) (5) (5) (10)

TXN (5) (5)

TXN (5) (40) (10) (35)

Question 4.

a) To represent the replay attack, I replicated (broadcasted again) the last transaction and showed it in the picture below with 2 red arrows.



b) In the transaction model this is avoided by having the inputs for each transaction pointing to some previous outputs. These outputs are now consumed for (sent to) different public keys, and the transaction must be signed by the private keys of each account for all the inputs. So, not only we now need to get the transaction signed, but most importantly the output of a transaction cannot be sent to the same public keys.

Question 5.

Transactions need to be signed by the private keys of the inputs' accounts owners because only then the transaction is considered willingly and can be validated. In other words, these signatures provide a proof of the consent of the senders of the funds.

Lab 2.

Task 1.

```
 *  Task 1: Fill in the method checkLedgerCanBeDeducted()
 *          It has been commented out so that the code compiles.
 *
 * Check all items in a ledger can be deducted from the current one
 *
 *   the ledger to be deducted is usually obtained
 *   from a list of inputs of a transaction
 */

// Checks if a ledger can be deducted.
public boolean checkLedgerCanBeDeducted(Ledger ledger2){
    //this.ledger is the current ledger.
    //ledger 2 is an AmountList (in this case a list with sums of amounts to
    //be subtracted from the current list,that can only be done
    //if they are less than the sums in the current ledger for each user.
     for ( String account : ledger2.getUsers()) {
         if (ledger2.getBalance(account) > this.getBalance(account)) {
             return false;
         }

     }
     return true;
};
```

Task 2.

```java
/** Task 2: Fill in the method checkUALCanBeDeducted
 *          It has been commented out so that the code compiles.
 *  It checks that a list of user amounts can be deducted from the
 *     current ledger
 *   done by first converting the list of user amounts into a ledger
 *     and then checking that the resulting ledger can be deducted.
 */
public boolean checkUALCanBeDeducted(UserAmountList ual){
    //ual is the resulting ledger.
    //this.ledger is the current ledger.
    //This method checks if the result of whatever transaction is possible by:
    //1. Checking if all the resulting balances are at least 0.
    //2. The sum of the current ledger is equal with the sum of the resulting ledger.
    //Or check directly that this ual can be subtracted from the current ledger,
    //by just checking that after the subtraction commits, for each user his balance is >=0.
    Ledger a=ual.toLedger();
    for ( String account : a.getUsers()) {
        if (this.getBalance(account) - a.getBalance(account)  < 0) {
            return false;
        }
    }
    return true;
};
```

Task 3.

```java
/**
 *  Task 3: Fill in the methods subtractUAL and  addUAL.
 * Subtract a list of user amounts (UAL) from the ledger
 *   requires that the list to be deducted is deducable.
 */
// Subtract a list of user amounts (UAL) from the ledger.
public void subtractUAL(UserAmountList ual){
    Ledger a=ual.toLedger();
    for ( String account : a.getUsers()) {
        this.setBalance(account, this.getBalance(account) - a.getBalance(account)  ) ;
    }
}

/**
 * Subtract a list of user amounts (UAL) to the current ledger
 */
// Add a list of user amounts (UAL) to the ledger.
public void addUAL(UserAmountList ual){
    Ledger a=ual.toLedger();
    for ( String account : a.getUsers()) {
        this.setBalance(account, this.getBalance(account) + a.getBalance(account)  ) ;
    }
}
```

Task 4.

```java
/**
 *
 *  Task 4: Fill in the method checkTransactionValid
 *          It has been commented out so that the code compiles.
 *
 * Check a transaction is valid:
 *    the sum of outputs is less than the sum of inputs
 *    and the inputs can be deducted from the ledger.
 *
 */

// Check a transaction is valid if:
// The sum of outputs is less than the sum of inputs
// The inputs can be deducted from the ledger.
public boolean checkTransactionValid(Transaction tr){
    return tr.checkTransactionAmountsValid()&&this.checkUALCanBeDeducted(tr.toInputs());
};
```

Task 5.

```java
/**
 *
 *  Task 5: Fill in the method processTransaction
 *
 * Process a transaction
 *    by first deducing all the inputs
 *    and then adding all the outputs.
 *
 */

// Process a transaction if valid by:
// First deducing all the inputs.
// Then adding all the outputs.
public void processTransaction(Transaction tr){

    if(this.checkTransactionValid(tr)) {
        this.subtractUAL(tr.toInputs());
        this.addUAL(tr.toOutputs());
    }
};
```

Task 6.

```java
Ledger c = new Ledger();
c.addAccount("Alice", 0);
c.addAccount("Bob", 0);
c.addAccount("Carol", 0);
c.addAccount("David", 0);
c.setBalance("Alice", 15);
c.setBalance("Bob", 10);
//Subtract 5 from balance of Bob
c.subtractFromBalance("Bob", 5);
c.print();

UserAmountList l1,l2;
//Check whether the user amount list giving Alice 10 units, and Alice again 10 units can be deducted.
l1=new UserAmountList("Alice",10,"Alice",10);
System.out.println(c.checkUALCanBeDeducted(l1));//false

//Check whether the user amount list giving Alice 10 units, and Bob 5 units can be deducted.
l2=new UserAmountList("Alice",10,"Bob",5);
System.out.println(c.checkUALCanBeDeducted(l2));//true
c.subtractUAL(l2);
c.addUAL(l1);
c.print();
```

```
<terminated> Ledger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw
The balance for Alice is 15
The balance for Bob is 5
The balance for Carol is 0
The balance for David is 0
false
true
The balance for Alice is 25
The balance for Bob is 0
The balance for Carol is 0
The balance for David is 0
```

```java
c.subtractFromBalance("Bob", 5);
c.print();

UserAmountList l1,l2;
//Check whether the user amount list giving Alice 10 units, and Alice agai
l1=new UserAmountList("Alice",10,"Alice",10);
System.out.println(c.checkUALCanBeDeducted(l1));//false

//Check whether the user amount list giving Alice 10 units, and Bob 5 unit
l2=new UserAmountList("Alice",10,"Bob",5);
System.out.println(c.checkUALCanBeDeducted(l2));//true
c.subtractUAL(l2);//deduct this list from the ledger
c.addUAL(l1);//add the list giving Alice twice 10 units to the ledger
c.print();

//Create a transaction which takes as input for Alice 30 units and gives Bob 5 and Carol 20 units.
Transaction tr=new Transaction(new UserAmountList("Alice",30),new UserAmountList("Carol",20,"Bob",5));
System.out.println(c.checkTransactionValid(tr));//Check whether it is valid = false

//Create a transaction which takes as input for Alice 20 units and gives Bob 5 and Carol 20 units.
Transaction tr1=new Transaction(new UserAmountList("Alice",20),new UserAmountList("Carol",20,"Bob",5));
System.out.println(c.checkTransactionValid(tr1));//Check whether it is valid = false

//Create a transaction which takes as input for Alice 20 units and gives Bob 5 and Carol 5 units.
Transaction tr2=new Transaction(new UserAmountList("Alice",20),new UserAmountList("Carol",5,"Bob",5));
System.out.println(c.checkTransactionValid(tr2));//Check whether it is valid = true
c.processTransaction(tr2);//Process transaction.
c.print();
```

```
<terminated> Ledger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\java
false
false
true
The balance for Alice is 5
The balance for Bob is 5
The balance for Carol is 5
The balance for David is 0  Until here
The balance for Alice is 0
The balance for Bob is 0
The balance for Carol is 10
```

```java
    c.addUAL(l1);//add the list giving Alice twice 10 units to the ledger
    c.print();

    //Create a transaction which takes as input for Alice 30 units and gives E
    Transaction tr=new Transaction(new UserAmountList("Alice",30),new UserAmou
    System.out.println(c.checkTransactionValid(tr));//Check whether it is vali

    //Create a transaction which takes as input for Alice 20 units and gives E
    Transaction tr1=new Transaction(new UserAmountList("Alice",20),new UserAmo
    System.out.println(c.checkTransactionValid(tr1));//Check whether it is val

    //Create a transaction which takes as input for Alice 20 units and gives E
    Transaction tr2=new Transaction(new UserAmountList("Alice",20),new UserAmountList("Carol",5,"Bob",5));
    System.out.println(c.checkTransactionValid(tr2));//Check whether it is valid = true
    c.processTransaction(tr2);//Process transaction.
    c.print();

    //Create a transaction which takes as input for Alice 5 units, Bob 5 units and gives Carol 5 and David 5 units.
    Transaction tr3=new Transaction(new UserAmountList("Alice",5,"Bob",5),
            new UserAmountList("Carol",5,"David",5));
    c.processTransaction(tr3);//Process transaction.
    c.print();
}
```

```
<terminated> Ledger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\java
true
The balance for Alice is 5
The balance for Bob is 5
The balance for Carol is 5
The balance for David is 0
The balance for Alice is 0
The balance for Bob is 0
The balance for Carol is 10
The balance for David is 5
```

From here

Lab 3.

Task 1.

Note: The function checkSignaturesValid1 is done by checking if the provided message (from which the signed message was generated in the examples) is valid. I wrote this function as I didn't understand at first if we need to generate the signatures for the transactions using the outputList class or use the signatures in the examples.

The function checkSignaturesValid2 does the proper check (where I create the signature for the message consisting of the output and input given by sender and amount).

```java
/* Task 1
   check all signatures are valid. In order for the code to compile it has been defined as True
   but that should be adapted. */
//checkSignaturesValid1 checks if the signature made from the initial provided message is valid.
public boolean checkSignaturesValid1() throws InvalidKeyException, NoSuchAlgorithmException, SignatureException{
    int size =  this.toInputs().size();
    ArrayList<Input> inputs = this.toInputs().toList();
    for(int i=0;i<size;i++) {
        Input inputEach=inputs.get(i);
        byte[] message=inputEach.getMessage();//Message provided in transaction parameters for each user.
        byte[] signature = inputEach.getSignature();//Signature provided in transaction parameters for each user.
        PublicKey puk=inputEach.getSender();
        if(!Crypto.verifySignature(puk, message,signature )) {
            return false;
        }
    }
    return true;
}
//checkSignaturesValid2 checks if the signature made from the outputList provided is valid.
public boolean checkSignaturesValid2(PublicPrivateKeyMap keyMap) throws InvalidKeyException, NoSuchAlgorithmException, SignatureEx
    int size =  this.toInputs().size();
    ArrayList<Input> inputs = this.toInputs().toList();
    OutputList outputs = this.toOutputs();
    for(int i=0;i<size;i++) {
        Input inputEach=inputs.get(i);
        byte[] signature = inputEach.getSignature();
        PublicKey puk=inputEach.getSender();
        byte[] message=outputs.getSignature(puk,inputEach.getAmount(),keyMap);//Signature generated by the outputList provided.
        if(!outputs.checkSignature(puk, inputEach.getAmount(),signature )) {
            return false;
        }
    }
    return true;
}
```

The following functions were added for the checkSignaturesValid1 to work:

In InputList class:

```java
public void addEntry(PublicKey sender,int amount,byte[] signature,byte[] message){
    inputList.add(new Input(sender,amount,signature,message));
}

/**
```

In Input class:

```java
public Input(PublicKey sender,int amount,byte[] signature,byte[] message){
    this.amount  = amount;
    this.sender = sender;
    this.signature = Arrays.copyOf(signature,signature.length);
    this.message=message;
}

public byte[] getMessage(){
    return message;
};
```

Task 2.

Note: checkSignaturesValid is the check where signatures are generated by output, input and amount, and checkSignaturesValid1 is the check where signatures are generated from messages in the example.

```java
/**
 *
 *  Task 2 Check a transaction is valid.
 *
 *  this means that
 *    the sum of outputs is less than the sum of inputs
 *    all signatures are valid
 *    and the inputs can be deducted from the ledger.

 *    the function has just been set to true so that the code compiles.
 * @throws SignatureException
 * @throws NoSuchAlgorithmException
 * @throws InvalidKeyException
 *
 */

public boolean checkTransactionValid(Transaction tr,PublicPrivateKeyMap keyMap) throws InvalidKeyException, NoSuchAlgorithmExcepti
    return tr.checkTransactionAmountsValid()&&tr.checkSignaturesValid2(keyMap)&&this.checkInputListCanBeDeducted(tr.toInputs());
};
```

```java
public boolean checkTransactionValid1(Transaction tr,PublicPrivateKeyMap keyMap) throws InvalidKeyException, NoSuchAlgorithmException, SignatureException{
    return tr.checkTransactionAmountsValid()&&tr.checkSignaturesValid1()&&this.checkInputListCanBeDeducted(tr.toInputs());
};
```

Task 3.

See code comments in the following images for explanations. All the code is submitted in the zip file (some of it is commented to allow for an easier choice which test or scenario to execute).

```java

    //signedMessage1 is sampleMessage1 of Alice signed
    //signedMessage2 is sampleMessage2 of Bob signed
    //Here I demonstrate that if the signature is not signed with the correct message it fails the check.
    tr = new Transaction(new InputList(pubKeyA,10,signedMessage1,sampleMessage1,pubKeyB,5,
                        signedMessage1,sampleMessage2),new OutputList(pubKeyA,7,pubKeyC,8));
        tr.testCase("Transaction Alice 10  Bob 5 to Alice 7 Carol 8",pubKeyMap);
        System.out.println("Transaction Alice 10  Bob 5 to Alice 7 Carol 8 regarding signature " +
                            tr.checkSignaturesValid1());

```

Console

```
<terminated> Transaction (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 6, 2019, 11:42:20 PM)
Transaction Alice 10  Bob 5 to Alice 7 Carol 8
Inputs:
User: Alice spends 10
User: Bob spends 5
Outputs:
User: Alice receives 7
User: Carol receives 8
Is valid regarding sums = true

Transaction Alice 10  Bob 5 to Alice 7 Carol 8 regarding signature false
```

```java

    /***   Task 3
               add  to the test case the example from Labsheet 1
               but now using the ledger based on public keys and signatures.
               In the part referring only to input annd output lists
               you can add the randomm signedMessages defined above as signatures.
               Once you have a transaction, you can define properly signed messages
               and check they are correctly signed.
               Create one example where an input is signed w.r.t. a different
               output list from the one being used.
    **/
    ledger.addAccount(pubKeyA, 20);
    ledger.addAccount(pubKeyB, 25);
    ledger.addAccount(pubKeyC, 40);
    ledger.print(pubKeyMap);
    //Check that the signature generated in the input of a transaction with the output, input and amount is valid.
    //The signature is signedMessage4.
    OutputList output4List = new OutputList(pubKeyB,10,pubKeyB,10);
    byte[] signedMessage4 = (new Input("Alice",20,output4List,keyMap)).getSignature();
    Transaction tr=new Transaction(new InputList(pubKeyA,20,signedMessage4),
            new OutputList(pubKeyB,10,pubKeyB,10));
    System.out.println("Transaction from Alice 20 to Bob 20 is " + tr.checkSignaturesValid2(keyMap));
    ledger.processTransaction(tr);
    ledger.print(pubKeyMap);
```

```
Console
<terminated> Ledger (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 12:03:57 AM)
Generating key pair for user Alice
Generating key pair for user Bob
Generating key pair for user Carol
Generating key pair for user David
Transaction from Alice 20 to Bob 20 is true
The balance for Carol is 40
The balance for Alice is 0
The balance for Bob is 45
```

Here is the scenario from lab 1 where transactions are generated with the messages from the examples.

```
eclipse projects2 - lab3Block/src/main/java/Ledger.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
TxLedger.java  Transaction.java  InputList.java  UTXOlist.java  TxOutput.java  Input.java  *Ledger.java  Transaction.java  InputList.java  OutputList.java  Input.java
```

```java
    //Here is the scenario from lab 1 where transactions are generated with the messages from the examples.
    Transaction tr1=new Transaction(new InputList(pubKeyA,15,signedMessage1,sampleMessage1),
            new OutputList(pubKeyB,5,pubKeyC,10));
    System.out.println(ledger.checkTransactionValid1(tr1,keyMap));//Transaction from Alice 15 to Bob 5 and Carol 10.
    ledger.processTransaction(tr1);
    ledger.print(pubKeyMap);

    System.out.println();
    ledger.addToBalance(pubKeyA, 5);//Transactions add to Alice 5.
    ledger.print(pubKeyMap);

    Transaction tr2=new Transaction(new InputList(pubKeyA,5,signedMessage1,sampleMessage1,pubKeyC,5,signedMessage3,sampleMessage3),
            new OutputList(pubKeyB,10));
    System.out.println(ledger.checkTransactionValid1(tr2,keyMap));//Transaction from Alice 5 and Carol 5 to Bob 10.
    ledger.processTransaction(tr2);
    ledger.print(pubKeyMap);
```

```
Console
<terminated> Ledger (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:20:22 AM)
Generating key pair for user Bob
Generating key pair for user Carol
Generating key pair for user David
The balance for Carol is 40
The balance for Bob is 25
The balance for Alice is 20
true
The balance for Carol is 50
The balance for Bob is 30
The balance for Alice is 5

The balance for Carol is 50
The balance for Bob is 30
The balance for Alice is 10
true
The balance for Carol is 45
The balance for Bob is 40
The balance for Alice is 5
Writable  Smart Insert  287 : 53
```

Here is the scenario from lab 1 where transactions are generated with the input, output and amount.

```java

    //Here is the scenario from lab 1 where transactions are generated with the input,output and amount.
    OutputList output4List = new OutputList(pubKeyB,5,pubKeyC,10);
    byte[] signedMessage4 = (new Input("Alice",15,output4List,keyMap)).getSignature();
    Transaction tr1=new Transaction(new InputList(pubKeyA,15,signedMessage4),
             new OutputList(pubKeyB,5,pubKeyC,10));
    System.out.println(ledger.checkTransactionValid(tr1,keyMap));//Transaction from Alice 15 to Bob 5 and Carol 10.
    ledger.processTransaction(tr1);
    ledger.print(pubKeyMap);

    System.out.println();
    ledger.addToBalance(pubKeyA, 5);//Transactions add to Alice 5.
    ledger.print(pubKeyMap);

    OutputList output5List = new OutputList(pubKeyB,10);
    byte[] signedMessage5 = (new Input("Alice",5,output5List,keyMap)).getSignature();
    byte[] signedMessage6 = (new Input("Carol",5,output5List,keyMap)).getSignature();
    Transaction tr2=new Transaction(new InputList(pubKeyA,5,signedMessage5,pubKeyC,5,signedMessage6),
             new OutputList(pubKeyB,10));
    System.out.println(ledger.checkTransactionValid(tr2,keyMap));//Transaction from Alice 5 and Carol 5 to Bob 10.
    ledger.processTransaction(tr2);
    ledger.print(pubKeyMap);

```

Console

```
<terminated> Ledger (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:33:46 AM)
Generating key pair for user Alice
Generating key pair for user Bob
Generating key pair for user Carol
Generating key pair for user David
The balance for Carol is 40
The balance for Alice is 20
The balance for Bob is 25
true
The balance for Carol is 50
The balance for Alice is 5
The balance for Bob is 30
```

Continuation of the console execution from above image:

```
The balance for Carol is 50
The balance for Alice is 5
The balance for Bob is 30

The balance for Carol is 50
The balance for Alice is 10
The balance for Bob is 30
true
The balance for Carol is 45
The balance for Alice is 5
The balance for Bob is 40
```

Lab 4.

Task 1.

```java
/**
 * Task 1.1
 * Add a check that the
 * sum of inputs is greater or equal than the sum of outputs
 *
 * however if a transaction is a coinase transaction (i.e. the
 *   list of inputs is empty
 *  then this check needs to return true
 *
 *  otherwise there would be no valid coinbase transactions
 *   and therefore it would not be possible to create any valid transactions
 *
 *
 *  The argument txLedger is only used in order to be able to printout readable
 *   information about which input has the error in case of an error.
 *  The hasMapLookupException is as well triggered by printing out this message.
 *
 */

public boolean checkTransactionAmountsValid (TxLedger txLedger)
throws HashMapLookupException {
    if (!this.isCoinbase()) {
        if(this.inputList().toSum(txLedger.utxoList()) < this.outputList().toSum()) {
            System.out.println("task1 fail");
            return false;
        }
    }
    return true;
}
```

```java
/**
 * Task 1.2
 * Check that the signatures in the inputs of a transaction are valid
 * Use txLedger and pubKeymap to print out the erroneous input in a readable format
 *
 * Again txLedger and pubKeyMap are used for printing out suitable error messages whereas
 */

public boolean checkSignaturesValid(TxLedger txLedger, PublicKeyMap pubKeyMap)
throws HashMapLookupException {
    int size =  this.inputList().size();
    ArrayList<Input> inputs = this.inputList().inputList();
    OutputList outputs = this.outputList();
    for(int i=0;i<size;i++) {
        Input inputEach=inputs.get(i);
        byte[] signature = inputEach.signature();
        PublicKey puk=inputEach.publicKey();
        //1. inputs of the transaction
        //2. pub key of each input
        //3. signature introduced by each input only in this transaction
        //4. those are checked by the function in outputList checkSignature
        if(!outputs.checkSignature(inputEach.txOutput(),puk,signature )) {
            System.out.println("task2 fail");
            return false;
        }
    }
    return true;


}

/*
 *
 * Task 1.3
 *  check all inputs in a transaction are unspent transaction outputs, i.e. in the underlying utxoList
 *  Use txLedger and pubKeymap to print out the erroneous input in a readable format
 *
 * Again txLedger and pubKeyMap are used for printing out suitable error messages whereas
 *
 */

//Check that each input in the transaction(txOutput for each input) is an entry in current utoxList.
public boolean checkInputsAreInUTXO(TxLedger txLedger,PublicKeyMap pubKeyMap)
throws HashMapLookupException
{
    int size =  this.inputList().size();
    ArrayList<Input> inputs = this.inputList().inputList();
    for(int i=0;i<size;i++) {
        Input inputEach=inputs.get(i);
        PublicKey puk=inputEach.publicKey();
        if(!txLedger.utxoList().hasEntry(inputEach.txOutput())) {
            System.out.println("task3 fail");
            return false;
        }
    }
    return true;
}
```

```java
/*
 *  Task 1.4
 *   check the inputs in the input list are different
 *   any transaction input can be used only once
 *   Use txLedger and pubKeymap to print out the erroneous input in a readable format
 *
 *   Again txLedger and pubKeyMap are used for printing out suitable error messages whereas
 *
 *   One way of programming this is to have an auxiliay set of transactiton outputs, which
 *      is initially empty. You can use HashSet to implement it.
 *   Then one goes through the transaction inputs one by one. If the transaction input is in the
 *     auxiliary set, then it occurs twice, otherwise one adds it to the set, so when it occurs
 *     again it is noticed.
 */
public boolean checkInputsDifferent(TxLedger txLedger,PublicKeyMap pubKeyMap)
throws HashMapLookupException
{
    //inputListFrequency is an empty ArrayList that memorizes the occurrences of each txOutput, to avoid duplications.
    ArrayList<TxOutput> inputListFrequency=new ArrayList<TxOutput>();
    int size =  this.inputList().size();
    ArrayList<Input> inputs = this.inputList().inputList();
    for(int i=0;i<size;i++) {
        if(inputListFrequency.contains(
                inputs.get(i).txOutput())) {
            System.out.println("task4 fail");
            return false;
        }
        inputListFrequency.add(inputs.get(i).txOutput());
    }
    return true;
}
```

```java
/*
 * Task 1.5
 *  Create a check function running all 4 checks above
 *
 * Again txLedger and pubKeyMap are used for printing out suitable error messages whereas
 */

public boolean check(TxLedger txLedger,PublicKeyMap pubKeyMap)
throws HashMapLookupException
{
    if(this.checkInputsAreInUTXO(txLedger, pubKeyMap)&&this.checkInputsDifferent(txLedger, pubKeyMap)&&
            this.checkSignaturesValid(txLedger, pubKeyMap)&&this.checkTransactionAmountsValid(txLedger)) {
    return true;
    }
    return false;
}
```

Task 2.

```java
    /**
        Task 2: Create the following test cases (each test case should be run using the testCase method above):.
        Testcase 1: a transaction with no inputs and outputs;
        Testcase 2: a coinbase transaction giving Alice and Bub 10 units each.
        Testcase 3: a transaction with input the 1st output of the coinbase transaction,
                    giving Bob and Carol 5 units each.
        Testcase 4: a transaction combining the two outputs to Bob (in total 15) and giving the
                result to Carol
    **/
    //Testcase1
    OutputList outputList1 = new OutputList();
    InputList inputList1 = new InputList();
    String transactionName1 = " ";
    Transaction tr1 = new Transaction(inputList1,outputList1,transactionName1);
    txLedger.testCase("Transaction " + transactionName1,tr1,pubKeyMap);

    //Testcase2
    InputList inputList2 = new InputList();
    OutputList outputList2 = new OutputList("Alice1",10,"Bob1",10,pubKeyMap);
    String transactionName2 = "coinbase(Alice1-10,Bob1-10)";
    Transaction tr2 = new Transaction(inputList2,outputList2,transactionName2);
    txLedger.testCase("Transaction " + transactionName2,tr2,pubKeyMap);
```

Console

```
<terminated> TxLedger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:52:50 AM)
Executing test()
Transaction
Transaction is valid = true
After processesing ledger=
[]

Transaction coinbase(Alice1-10,Bob1-10)
Transaction is valid = true
After processesing ledger=
[(txoutput=coinbase(Alice1-10,Bob1-10)[1],recipient=Bob1,amount=10),
(txoutput=coinbase(Alice1-10,Bob1-10)[0],recipient=Alice1,amount=10)]
```

```java
    //Testcase3
    OutputList outputList3 = new OutputList("Carol1",5,"Bob2",5,pubKeyMap);
    InputList inputList3 = new InputList(transactionName2,0,outputList3,txLedger,keyMap);
    String transactionName3 = "Alice1-10->Carol1-5,Bob2-5";
    Transaction tr3 = new Transaction(inputList3,outputList3,transactionName3);
    txLedger.testCase("Transaction " + transactionName3,tr3,pubKeyMap);

    //Testcase4
    OutputList outputList4 = new OutputList("Carol2",15,pubKeyMap);
    Input inputList4a = new Input(transactionName2,1,outputList4,txLedger.utxoList(),txLedger.txIdNameMap,keyMap);
    Input inputList4b = new Input(transactionName3,1,outputList4,txLedger.utxoList(),txLedger.txIdNameMap,keyMap);
    InputList inputList4=new InputList(inputList4a,inputList4b);
    String transactionName4 = "Bob1-10,Bob2-5->Carol2-15";
    Transaction tr4 = new Transaction(inputList4,outputList4,transactionName4);
    txLedger.testCase("Transaction " + transactionName4,tr4,pubKeyMap);
```

Console

```
<terminated> TxLedger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:52:50 AM)
Transaction Alice1-10->Carol1-5,Bob2-5
Transaction is valid = true
After processesing ledger=
[(txoutput=coinbase(Alice1-10,Bob1-10)[1],recipient=Bob1,amount=10),
(txoutput=Alice1-10->Carol1-5,Bob2-5[1],recipient=Bob2,amount=5),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]

Transaction Bob1-10,Bob2-5->Carol2-15
Transaction is valid = true
After processesing ledger=
[(txoutput=Bob1-10,Bob2-5->Carol2-15[0],recipient=Carol2,amount=15),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]
```

Task 3.

```java
/** Task 3: Create for each check defined in Task 1 one test case, which
             will fail that particular check (and passes all checks coming before it)**/
//task1fail the output list has too big of values.
OutputList outputList5 = new OutputList("Carol1",105,"Bob2",5,pubKeyMap);
InputList inputList5 = new InputList(transactionName4,0,outputList5,
                    txLedger,keyMap);
String transactionName5 = "Carol2-15->Carol1-105,Bob2-5";
Transaction tr5 = new Transaction(inputList5,outputList5,transactionName5);
txLedger.testCase("Transaction " + transactionName5,tr5,pubKeyMap);

//task2fail the output which is used to generate signatures is different when generating input from when creating transactions.
OutputList outputList6 = new OutputList("Carol1",5,"Bob2",5,pubKeyMap);
InputList inputList6 = new InputList(transactionName4,0,outputList6,
                    txLedger,keyMap);
String transactionName6 = "Carol2-15->Carol1-5,Bob2-5";
Transaction tr6 = new Transaction(inputList6,outputList5,transactionName6);
txLedger.testCase("Transaction " + transactionName6,tr6,pubKeyMap);
```

Console

```
<terminated> TxLedger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:52:50 AM)

Transaction Carol2-15->Carol1-105,Bob2-5
task1 fail
Transaction is valid = false
task1 fail
After processesing ledger=
[(txoutput=Bob1-10,Bob2-5->Carol2-15[0],recipient=Carol2,amount=15),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]

Transaction Carol2-15->Carol1-5,Bob2-5
task2 fail
Transaction is valid = false
task2 fail
After processesing ledger=
[(txoutput=Bob1-10,Bob2-5->Carol2-15[0],recipient=Carol2,amount=15),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]
```

For task3fail I had to also comment the following line which would cause an error check in TxLedger class:

```java
if (PRINTBASE58)
    tr.printBase58();
//tr.print(utxoList(),txIdNameMap(),pubKeyMap);
System.out.println("Transaction is valid = " + checkTransaction(tr,pubKeyMap));
```

```java
//task3fail check if the txoutput 0 from transaction 2(finished before transaction4) is in current utoxlist (last transaction in utoxList is transaction4).
OutputList outputList7 = new OutputList("Carol1",5,"Bob2",5,pubKeyMap);
TxId txId = txLedger.txIdNameMap.name2TxId(transactionName2);
Input input7 = new Input(txId,0,outputList7,pubKeyA,keyMap);
String transactionName7 = "Alice1-10->Carol1-5,Bob2-5";
InputList inputList7 = new InputList(input7);
Transaction tr7 = new Transaction(inputList7,outputList7,transactionName7);
txLedger.testCase("Transaction " + transactionName7,tr7,pubKeyMap);

//task4fail The inputs are duplicated.
OutputList outputList8 = new OutputList("Carol1",5,pubKeyMap);
Input inputList8a = new Input(transactionName4,0,outputList8,txLedger.utxoList(),txLedger.txIdNameMap,keyMap);
Input inputList8b = new Input(transactionName4,0,outputList8,txLedger.utxoList(),txLedger.txIdNameMap,keyMap);
InputList inputList8=new InputList(inputList8a,inputList8b);
String transactionName8 = "Carol2-15->Carol1-5";
Transaction tr8 = new Transaction(inputList8,outputList8,transactionName8);
txLedger.testCase("Transaction " + transactionName8,tr8,pubKeyMap);
```

Console

```
<terminated> TxLedger [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (Nov 7, 2019, 1:52:50 AM)
Transaction Alice1-10->Carol1-5,Bob2-5
task3 fail
Transaction is valid = false
task3 fail
After processesing ledger=
[(txoutput=Bob1-10,Bob2-5->Carol2-15[0],recipient=Carol2,amount=15),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]

Transaction Carol2-15->Carol1-5
task4 fail
Transaction is valid = false
task4 fail
After processesing ledger=
[(txoutput=Bob1-10,Bob2-5->Carol2-15[0],recipient=Carol2,amount=15),
(txoutput=Alice1-10->Carol1-5,Bob2-5[0],recipient=Carol1,amount=5)]
```