**华东师范大学数据科学与工程学院实验报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：区块链与分享型数据库 | **年级**：21 | **上机实践成绩**： |
| **指导教师**：张召 | **姓名**：洪贝贝 | **学号**：10215501418 |
| **上机实践名称**：区块链系统(minichain)的简单实现 | | **上机实践日期**：2022.2.24 |
| **上机实践编号**： | **组号**： | **上机实践时间**：2024/3/7 |

1. **实验目的**

参考比特币中的区块结构，使用Java程序设计语言实现一个简单的区块链系统(minichain)，从工程角度学习并理解区块链中的各种概念。

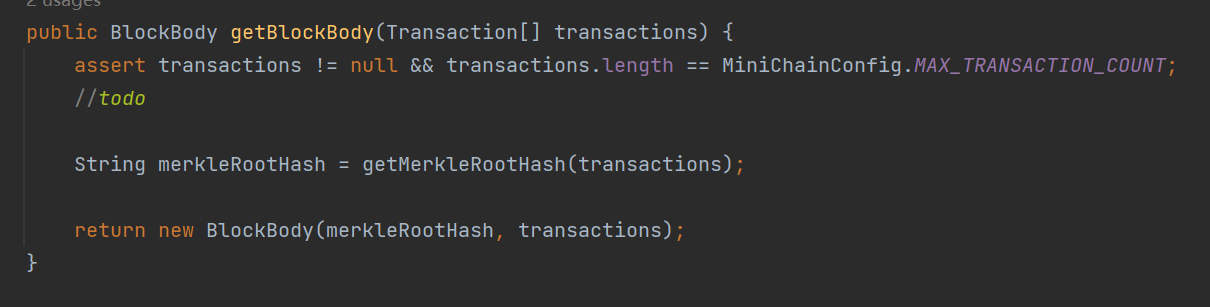
1. **实验任务**
2. 基本区块结构及其必要属性
3. 掌握Merkle树的基本原理，学会计算区块的哈希值
4. 实现基于POW共识的挖矿过程
5. 实现账户类与钱包功能
6. 实现UTXO类与解锁脚本功能
7. 实现账户签名交易，矿工验签功能
8. SPV 轻节点模块与验证
9. **实验环境**

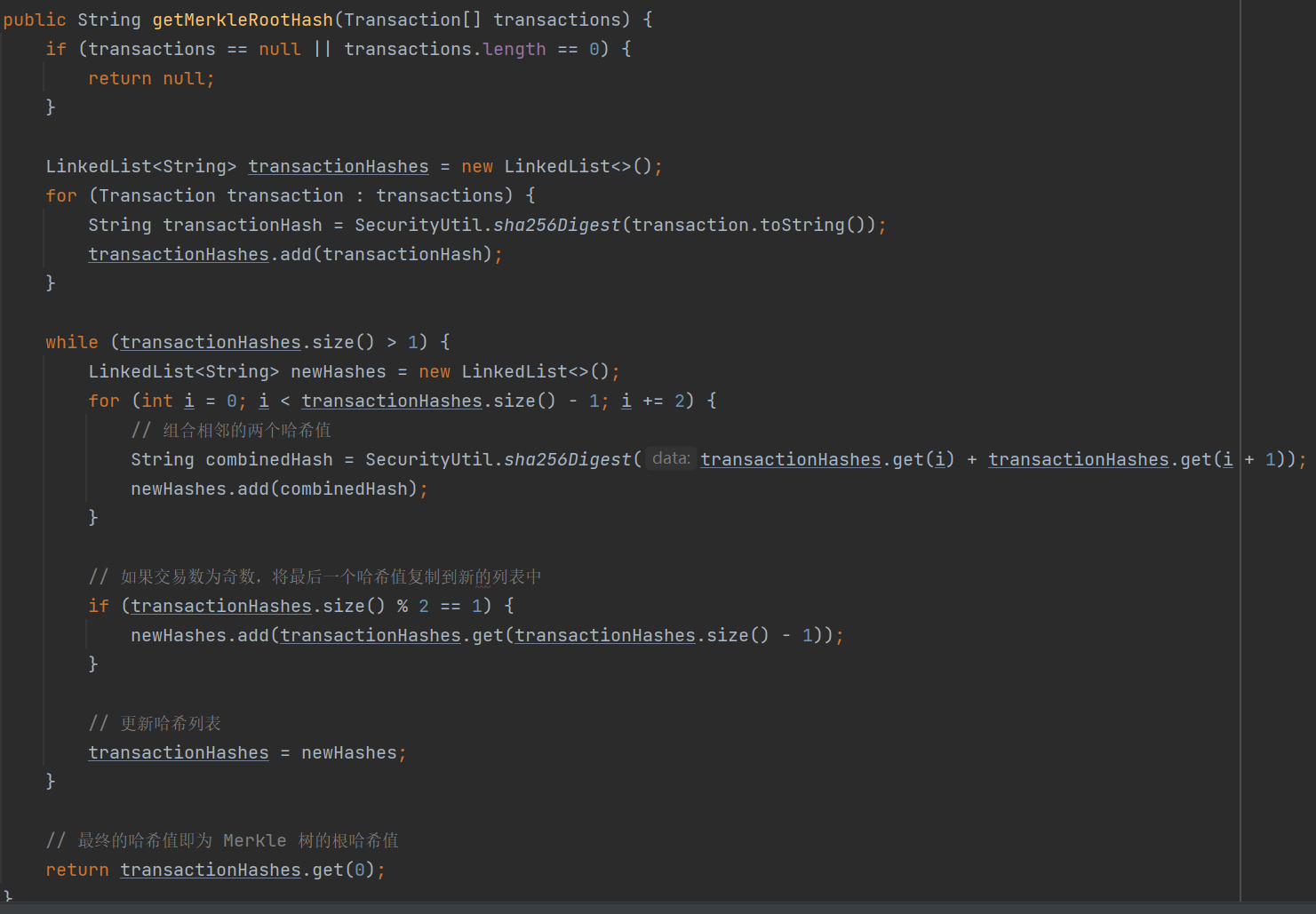
JDK:19/1.8(第一次小实验为19，第二、三次为1.8)

IntelliJ IDEA 2022.3.3

1. **实验过程**
   * + 1. 补全MinerNode中的几个功能函数
2. getBlockBody函数，该方法根据传入的参数中的交易，构造并返回一个相应的区块体对象，还需要根据这些交易计算Merkle树的根哈希值；

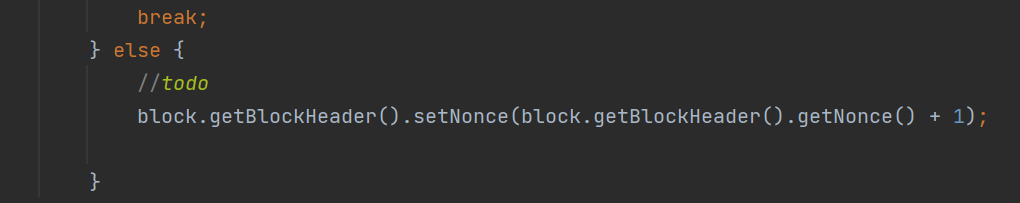
* 添加getMerkleRootHash以计算Merkle树的根哈希值





1. mine函数，在循环中完成"挖矿"操作，其实就是通过不断的变换区块中的nonce字段，直至区块的哈希值满足难度条件，即可将该区块加入区块链中；

* 通过+1改变区块中的nonce字段



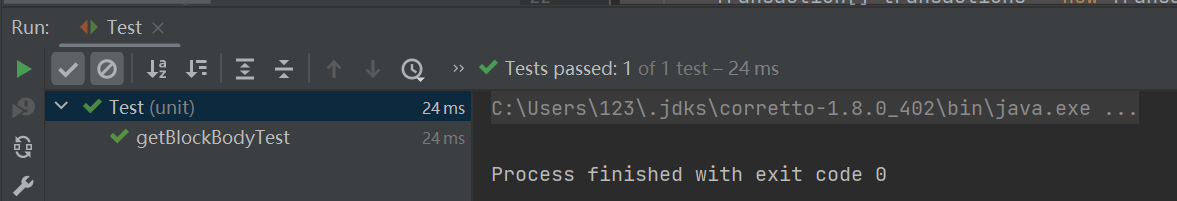
1. getBlock函数，该方法供mine方法调用，其功能为根据传入的区块体参数，构造一个区块对象返回。



* + - 1. 阶段一（第一次小实验）运行结果

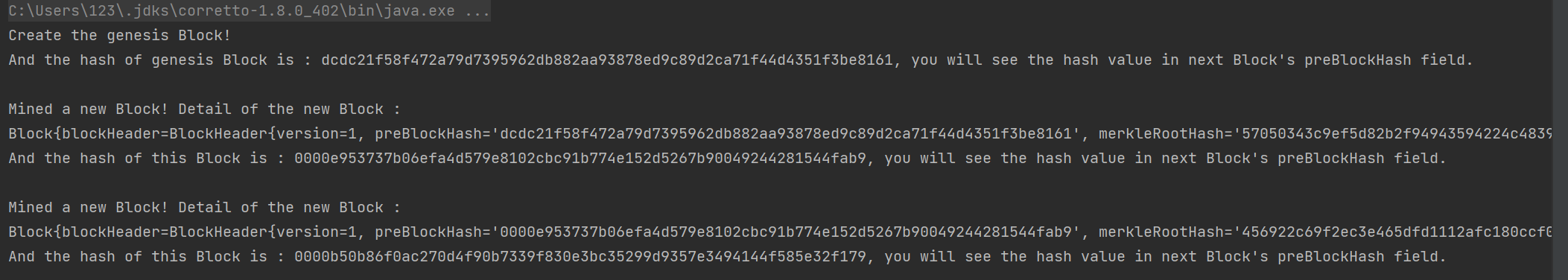
1. 运行getBlockBodyTest

* 测试结果通过：



1. 运行整个系统

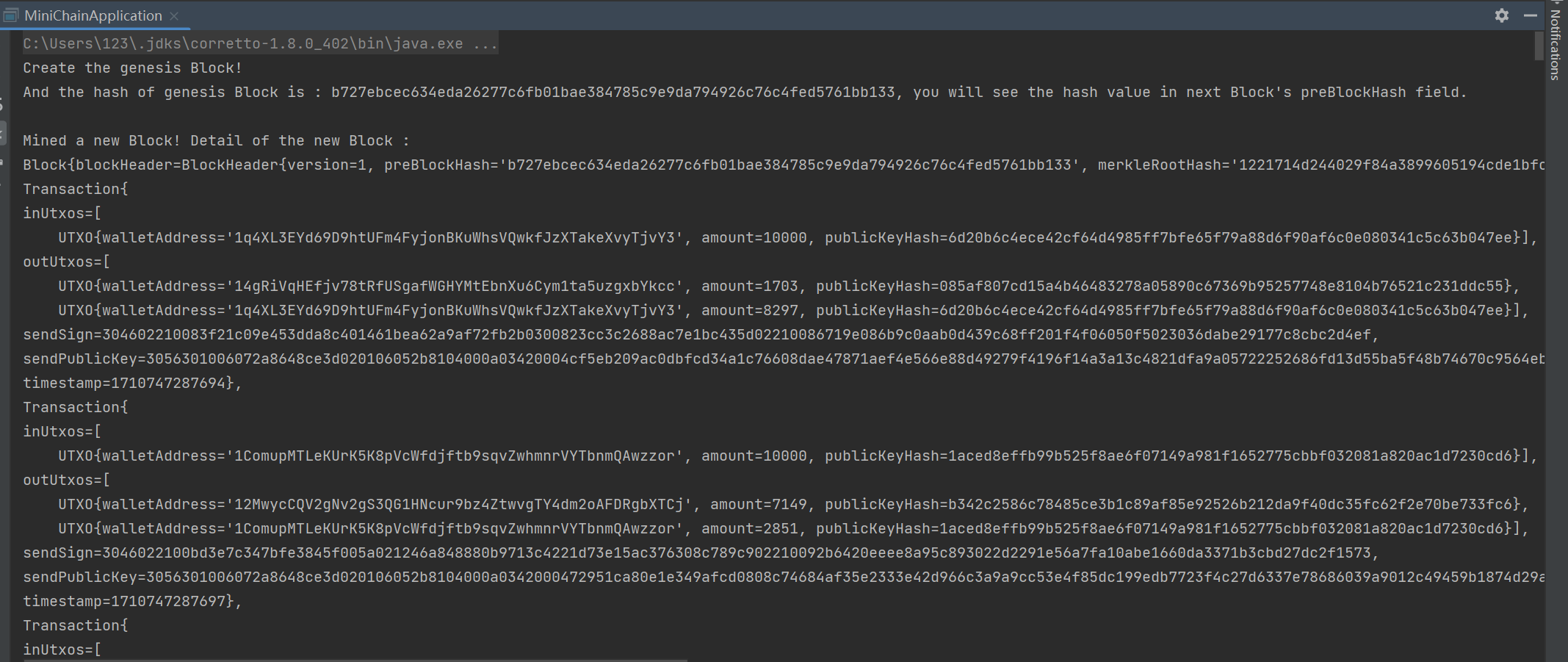
* 加入的区块的哈希值和挖掘到的区块的哈希值保持一致：



* + - 1. 按照实验指导手册进行代码补全及修改

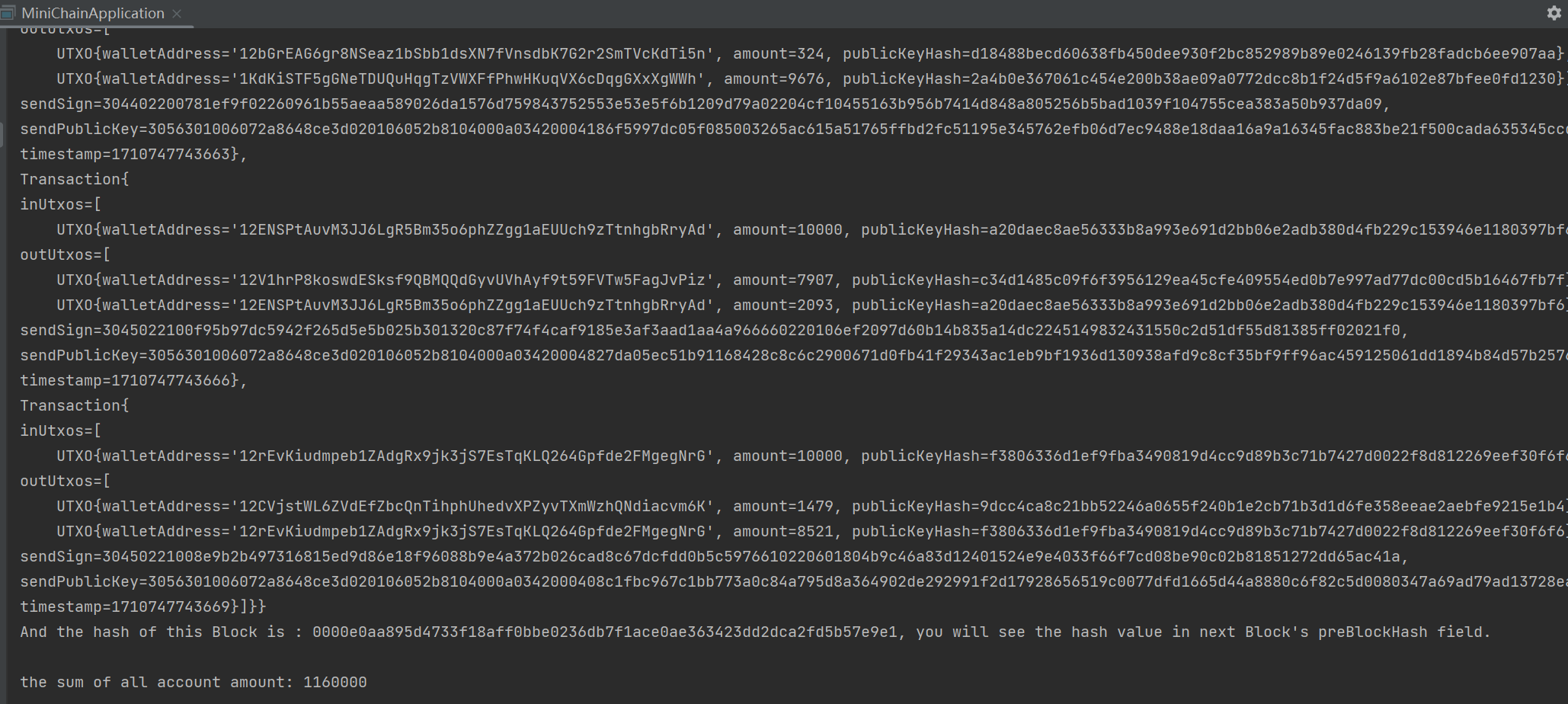
1. 修改Util工具类支持
2. 新建并完善Account类、UTXO类
3. 修改Transactions类、MiniChainConfig类、BlockChain类、TransactionProducer类、NetWork构造方法、MinerNode类
4. 将Test测试代码中因代码修改而出错的地方注释
   * + 1. 阶段二（第二次小实验）运行结果
5. 运行MiniChainApplication类中的main方法启动minichain

* 里面有若干个签名交易，交易中有若干输入UTXO和输出UXTO



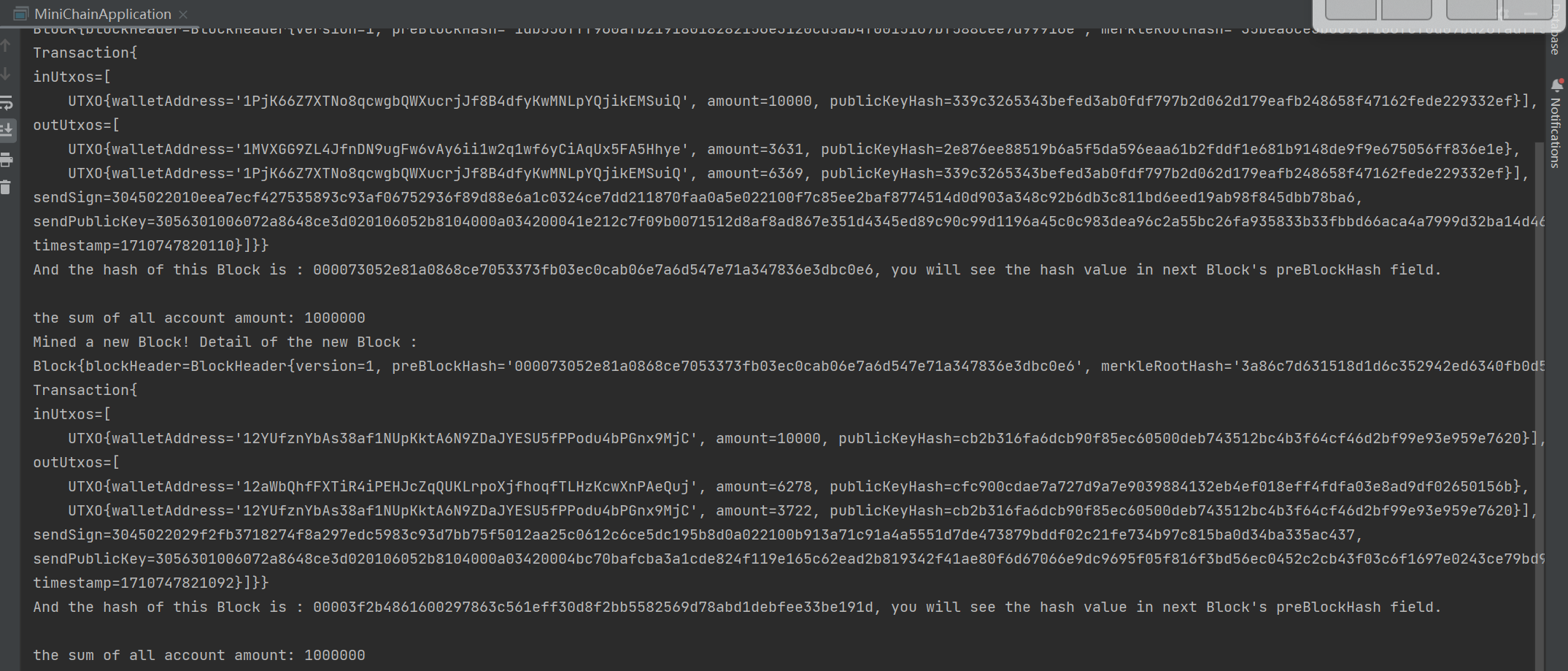
1. 在BlockChain类中添加统计下所有账户的总余额的方法,并在在MineNode类中的run方法添加输出代码，重新运行（1）中代码得到如下结果

* 总金额与预期的1000000不一样。这因为当前每个块的交易数是64个，有些交易使用了重复的输入UTXO，所以总金额会慢慢变大。



1. 将每个块的交易数限制为1个

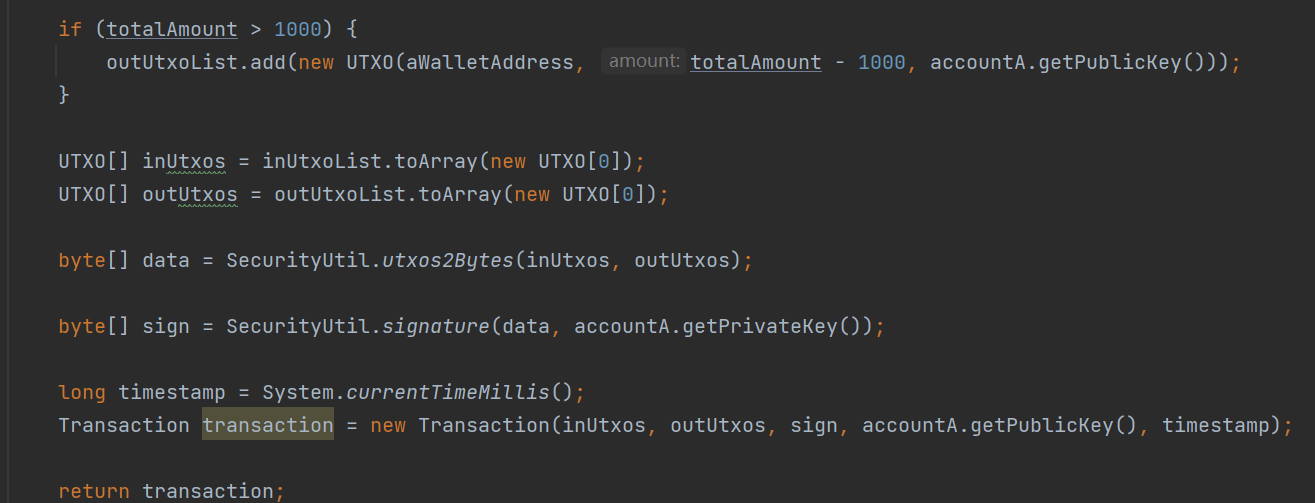
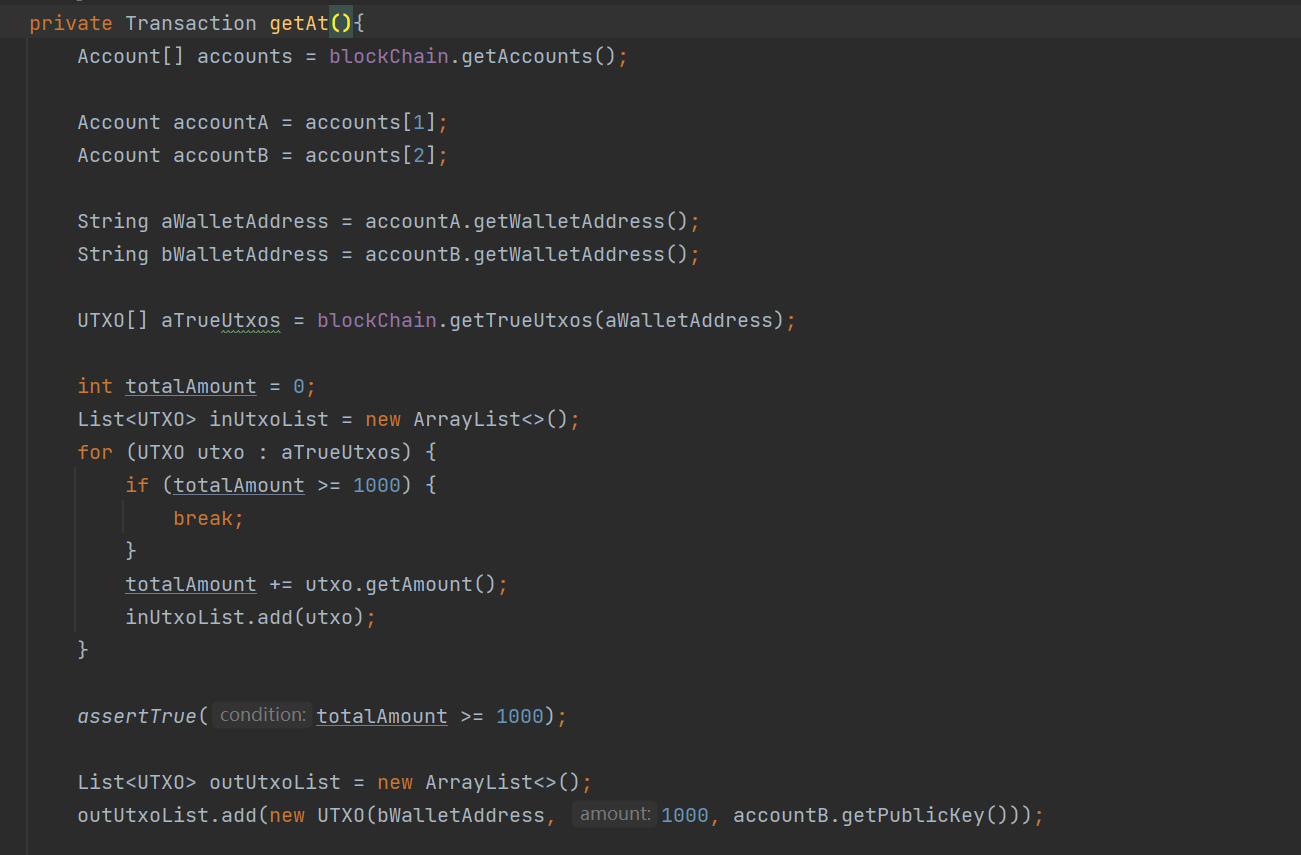
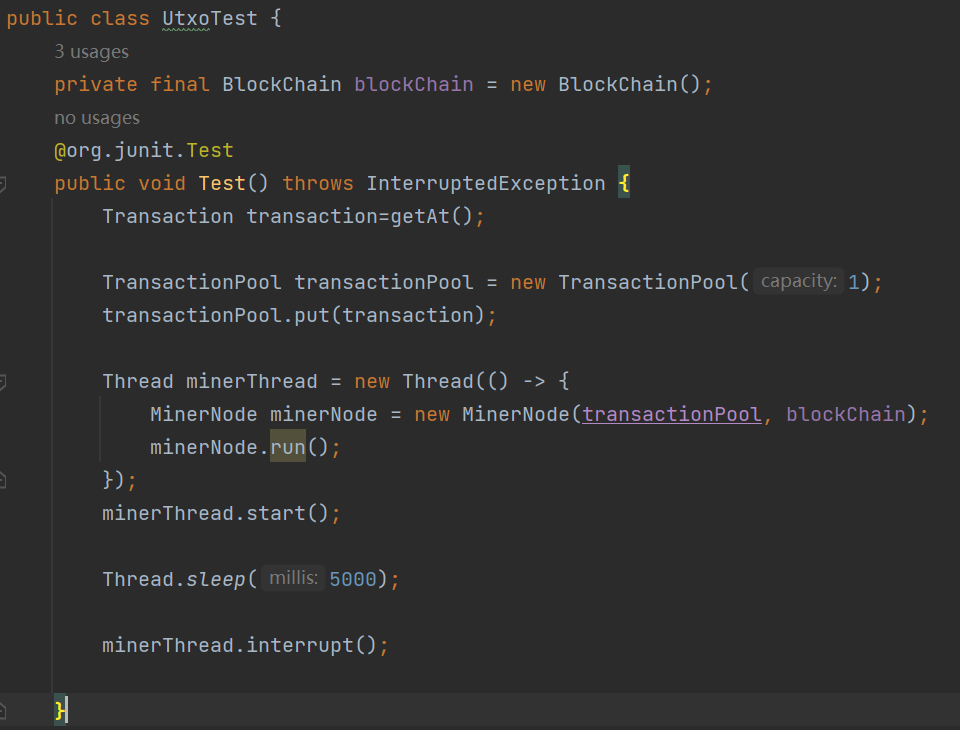
* 重新运行，所有账户的总余额为1000000，并且一个交易中的输入UTXO和输出UTXO在金额上时相等的



* + - 1. 在test/java/unit包中新建一个测试类UtxoTest，构造一个确定的UTXO交易：账户A(accounts[1])转账1000 BTC 给账户B(accounts[2])。

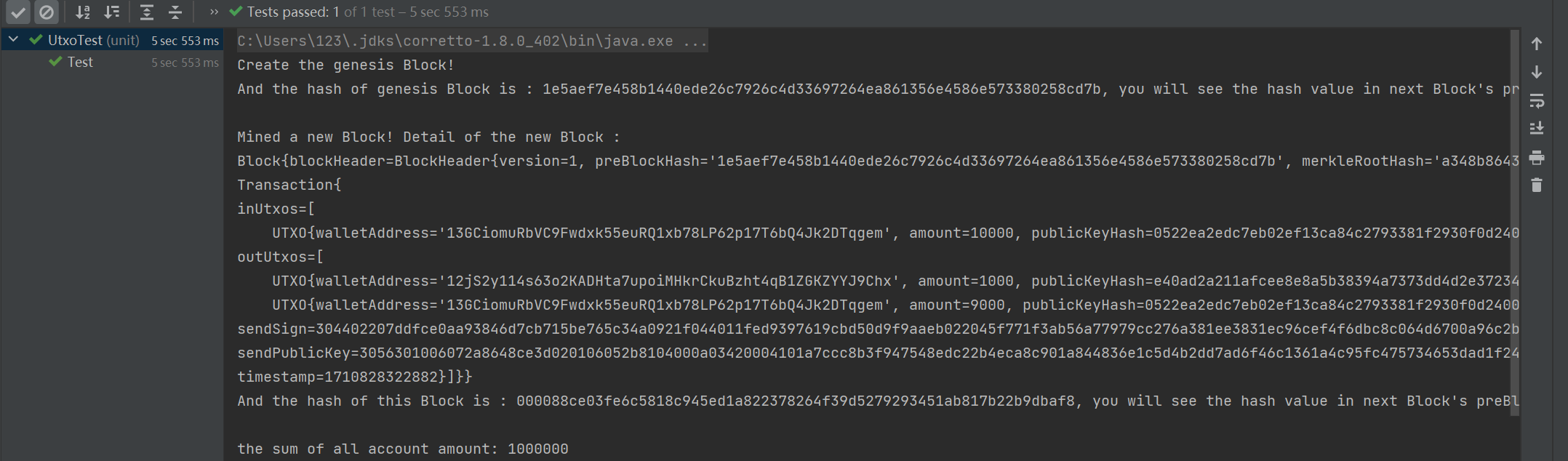
1. 代码实现

* 生成该交易的过程与TransactionProducer类中的getOneTransaction()类似，但是交易金额变成了固定的1000。



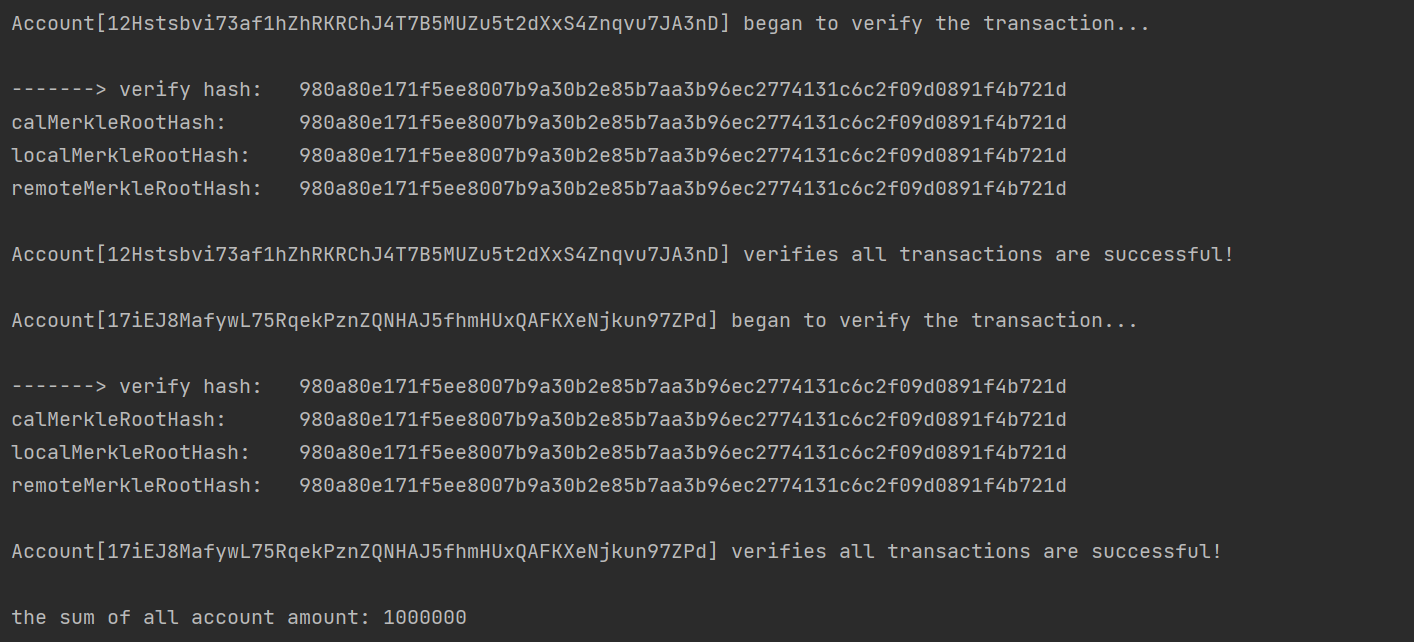
1. 测试运行结果

* 成功实现一笔交易使得a转给b 1000BTC，通过 minerThread.interrupt() 方法中断了矿工节点的线程,不然这个测试停不下来，因为矿工节点一直在尝试挖矿



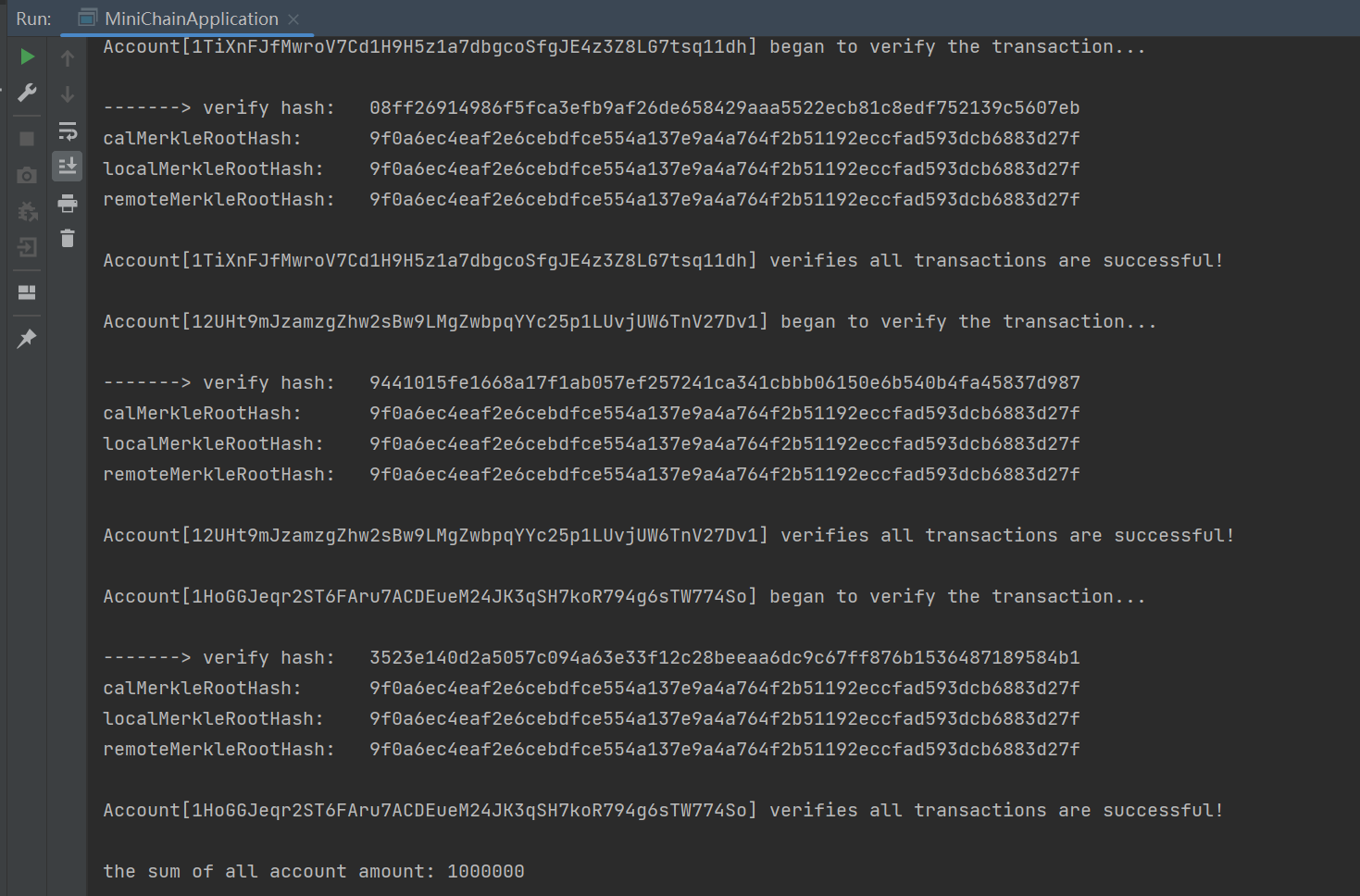
* + - 1. 按照实验指导手册进行代码补全及修改(重构之前的代码框架)

1. 使用网络模块作为其他模块通信的中介
2. 初始产生utxo的函数由BlockChain类迁移至Network类
3. 检查放入池中的交易是否已经有池中交易使用的UTXO，如果有，则拒绝该交易入池(要注意修改区块交易数，如果未将其从1改为其他数字，后面产生的txHash和本地的Merkle根哈希和从矿工节点返回的根哈希以及对返回的验证路径验证后的根哈希四个都是一样的)

****

1. 添加SpvPeer类、Proof类、simplifiedPaymentVerify函数、getProof函数
2. 在网络中加入spv轻节点，并将其关联至账户
3. 添加广播区块头的相关内容
4. 添加验证的相关代码
   * + 1. 阶段三（第三次小实验）运行结果

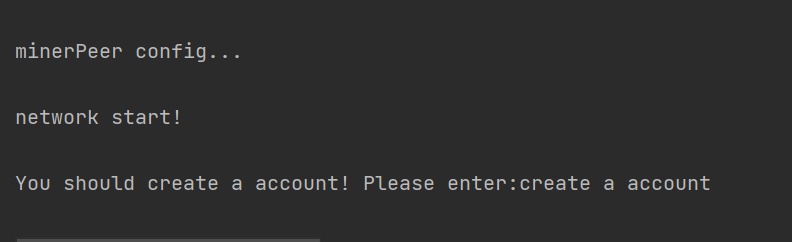
* 运行主程序， 得到本地的Merkle根哈希和从矿工节点返回的根哈希以及对返回的验证路径验证后的根哈希三者是一样的，验证成功。

****

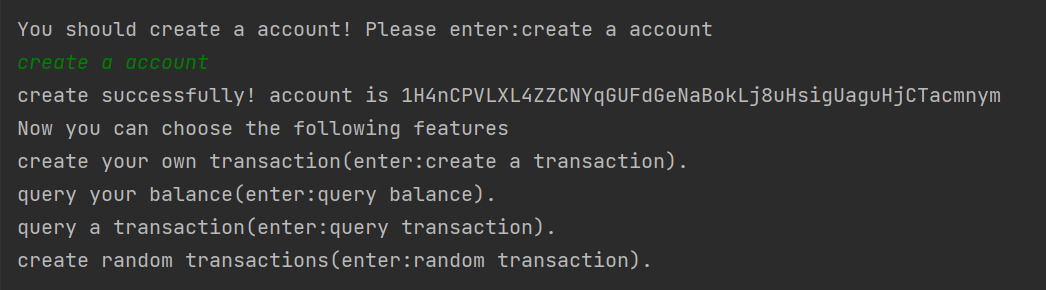
* + - 1. 阶段三（第三次小实验）拓展内容

1. 合并TransactionProducer和SpvPeer，重构network类
2. 创建账户，创建转账交易, 查询账户余额, 验证交易, 创建随机交易:通过在终端键盘输入来模拟用户与钱包客户端的交互
3. 构造create\_account()函数, getbalance()函数
4. 运行结果

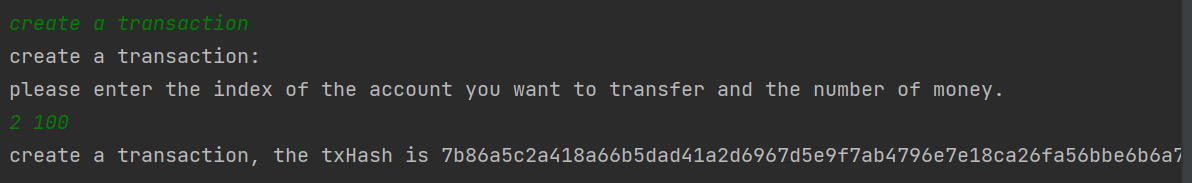
* 运行程序，初试界面如下：

****

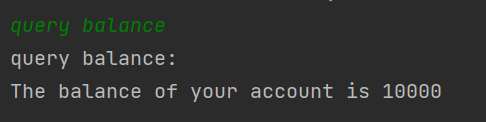
* 终端键盘输入create a account

****

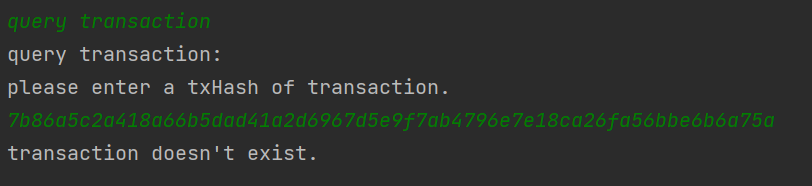
* 终端键盘输入create a transaction

****

* 终端键盘输入query balance ,查询账户余额，但是余额没有改变，因为交易还在交易池中没有被确认上链

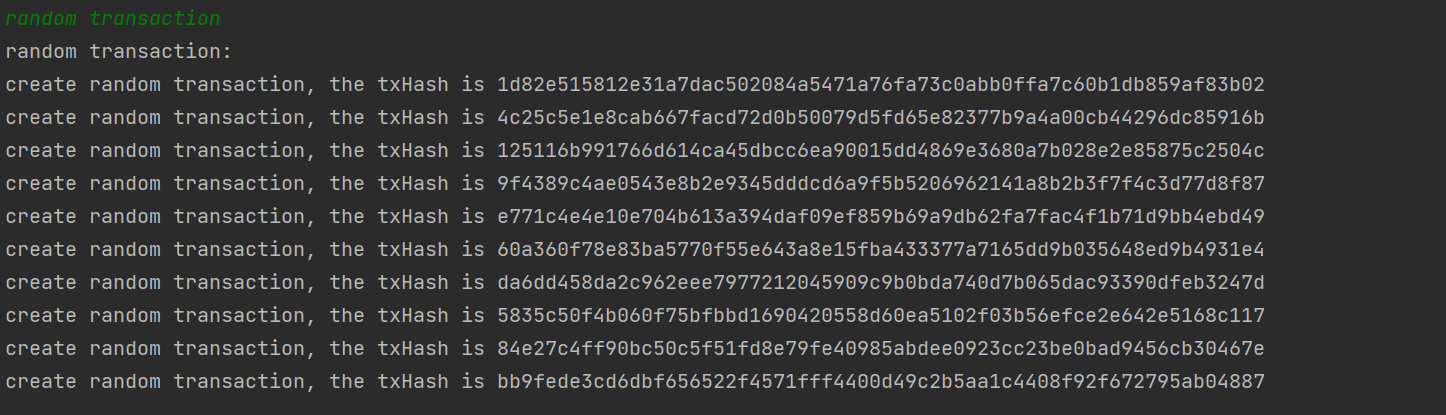
****

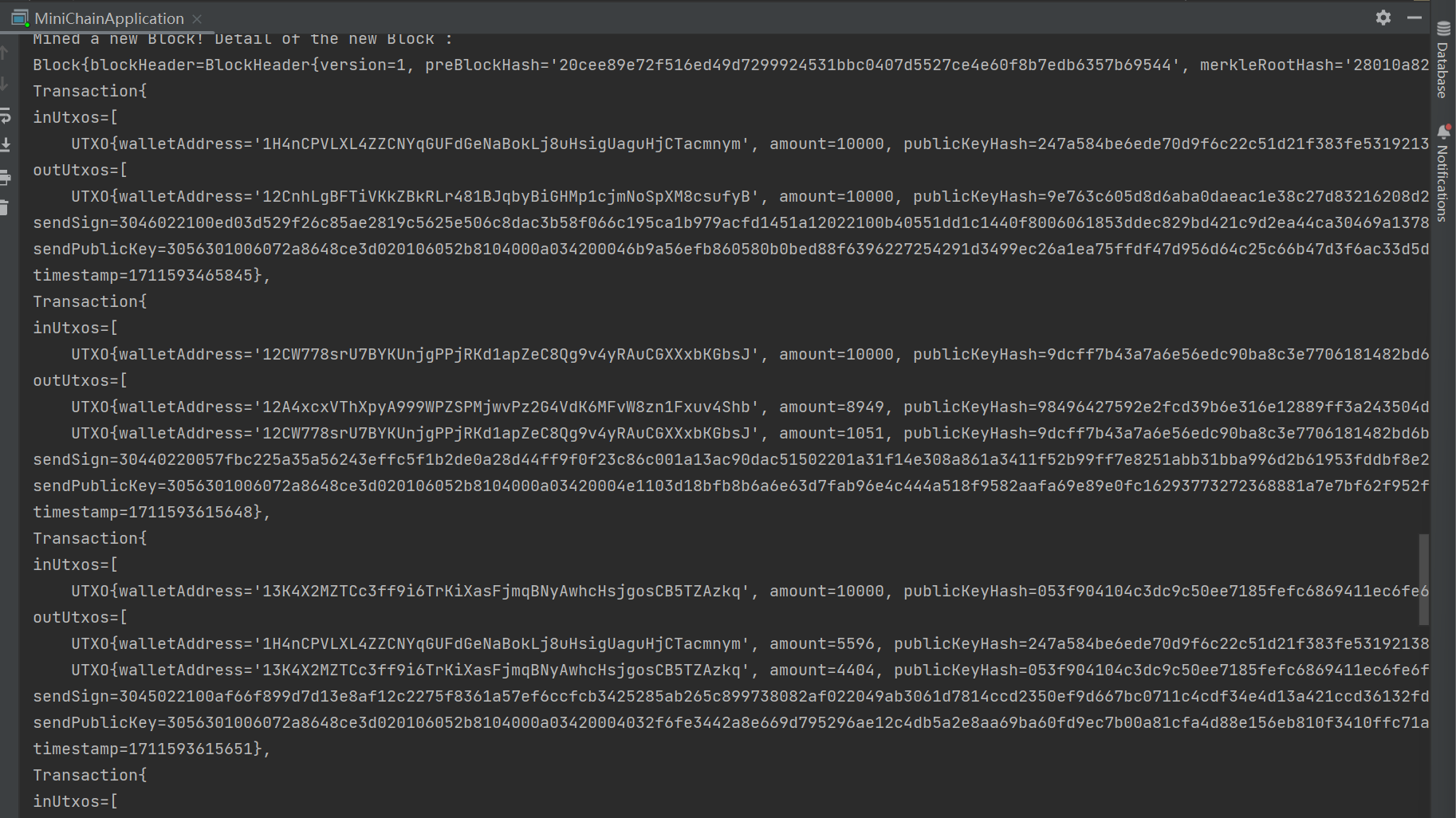
* 终端键盘输入query transaction并输入上述的txHash，发现网络返回的交易不存在。

****

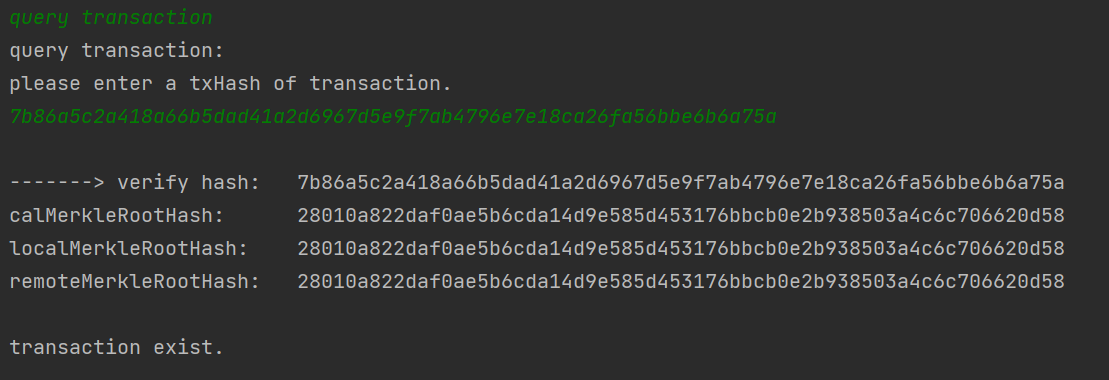
* 为了使得交易上链，我们终端键盘输入random transaction，生成新的区块。
* 为什么我们不能再通过该账户创建消息来使得交易池填满？

如果之前的交易尚未被确认上链，那么相应账户的nonce值可能已经被消耗了。因此，新的交易可能会被视为无效，因为nonce值不匹配。

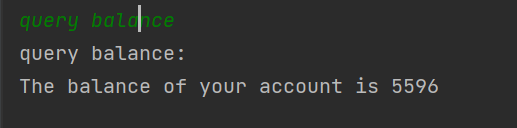
****

****

* 终端键盘输入query transaction并输入上述的txHash，返回交易存在

****

* 终端键盘输入query balance ,查询账户余额，账户余额正确

****

1. **习题**

1. 现有一Mekle树如下图所示，请简述树根（Top Hash）的形成过程。

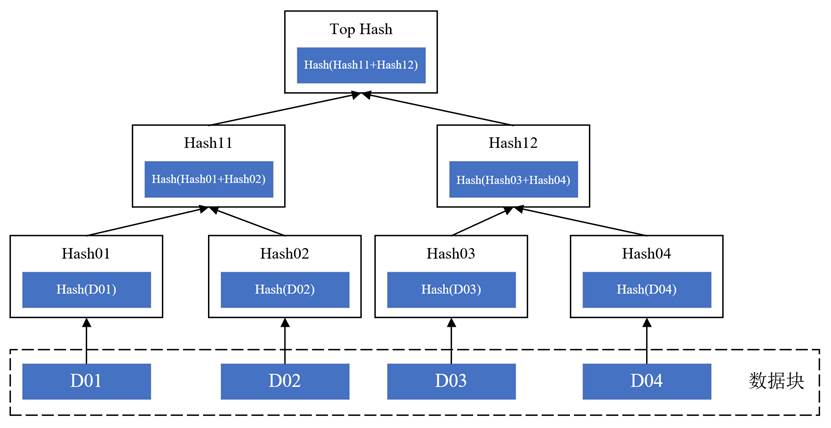


图 1-6 树根形成过程

1. 将数据块 D01、D02、D03、D04 分别进行哈希运算，得到它们的哈希值 Hash01=Hash(D01)、Hash02=Hash(D02)、Hash03=Hash(D03)、Hash04=Hash(D04)。
2. 将相邻的叶子节点的哈希值两两合并，并再次进行哈希运算。这样得到的新的哈希值就是它们的父节点的哈希值。即为Hash11=Hash(Hash01  + Hash02) 和 Hash12=Hash(Hash03  + Hash04)。
3. 将两个中间节点的哈希值再次合并，得到最终的树根（Top Hash）的哈希值。在这个例子中，最终的树根哈希值为Top Hash=Hash(Hash11  + Hash12)。

2. 现有数据A~E，其值分别为A=4，B=3，C=1，D=5，E=8，采用哈希函数hash(x) = x%5，试求由这些数据构成的Merkle树的根哈希。并在图中绘制出Merkle树生成的过程。

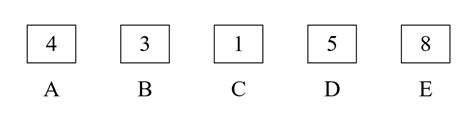


图 1-7 Merkle树原始数据块

根哈希值为0

3. 简述本实验代码中挖矿的过程。

挖矿的过程是不断地从交易池中获取交易，

通过 `getBlockBody` 接收一批交易作为参数，并根据这些交易计算 Merkle 树的根哈希值，并构造一个包含该根哈希值和交易数组的区块体对象。

并调用 `mine` 方法持续不断地尝试不同的随机字段（nonce），直到找到一个合适的值，使得区块的哈希值满足难度条件。

然后通过`getBlock`方法获取最新的区块，然后根据该区块的哈希作为前一个区块的哈希，使用前面随机生成的 nonce 构造一个新的区块头，并将区块头和区块体组合成一个完整的区块对象，然后添加到区块链中。

1. 在非对称加密算法和数字签名的过程中，私钥和公钥的作用是否相同？若不同，请描述不同之处。

(1) 非对称加密算法：

在加密过程中，数据使用接收者的公钥进行加密，只有使用接收者的私钥才能解密数据。所以私钥用于解密数据，公钥用于加密数据

(2) 数字签名：

发送者使用私钥对数据的摘要（或哈希）进行签名，生成数字签名。接收者使用发送者的公钥对接收到的数据和数字签名进行验证，以确保数据的完整性和来源的真实性。所以私钥用于签名生成，公钥用于验证签名。

2. UTXO模型和账户模型的各自优缺点是什么？

(1) UTXO 模型优缺点：

**优点：**

UTXO 模型隐私性更高，因为它不直接暴露账户余额和交易历史。每个 UTXO 都对应着一个交易输出，这使得跟踪特定用户的交易历史变得更加困难。

UTXO 模型可以同时处理多个UTXO，因此可以实现并行事务并鼓励可伸缩性创新。

**缺点：**

因为每个交易都需要确保消费的 UTXO 是有效的，并且需要处理找零（Change）等问题。所以编程复杂性较高

每个交易可能会包含多个输入和输出，比较消耗CPU和存储空间。

(2)账户模型优缺点：

**优点：**

账户模型更加简单直观，每个账户直接对应着一个余额，容易理解和管理。

交易效率高，因为每笔交易只需要验证发送账户是否有足够的余额来支付交易。

**缺点：**

账户模型的交易并行性较差，因为不同账户之间的交易可能会有竞争条件导致的冲突。

* + - 1. 为什么要使用SPV简单支付验证？

SPV客户端只需下载并验证与自己相关的区块头，而不需要存储整个区块链的副本。

1. 可以节约资源
2. 泄露的隐私更少
3. 可以更快地开始验证交易
   * + 1. 为什么SPV简单支付验证可以保证其准确性（从其应用节点存放的数据结构和验证过程来说明）？

1.数据结构：

SPV客户端存储的数据结构通常包括区块头，而不是完整的区块链。区块头包含了每个区块的摘要信息，包括区块的哈希、时间戳、难度目标以及Merkle根等。这些信息可以使得SPV客户端验证某个区块是否被网络中的大多数节点所接受。

2. 验证过程：

- 验证区块头：SPV客户端通过验证区块头来确保某个区块的有效性。它可以检查区块头中的哈希值是否满足网络中的难度目标，以及该区块的时间戳是否合理。这些验证步骤可以保证区块的工作量证明（Proof of Work）是有效的，从而增强了区块链的整体安全性。

- 验证交易：虽然SPV客户端不验证所有交易，但它们可以通过请求网络中的一些完整节点来验证特定交易是否被包含在了一个已经验证的区块中。这种验证过程使得客户端能够确定自己的交易是否被网络所接受。

* + - 1. 全节点有没有可能既欺骗了我们，又保证最终计算的merkle根哈希与区块头中相同呢？

全节点在保证最终计算的Merkle根哈希与区块头中相同的情况下，同时欺骗我们是不太可能的。Merkle根哈希是通过对区块中所有交易的哈希进行递归计算而得到的，因此要同时欺骗并保证Merkle根哈希与区块头中相同是非常困难的。且区块头中的Merkle根哈希是由其他节点进行验证的重要部分。其他节点可以通过请求区块中的交易数据，并自行计算Merkle根哈希来验证它是否与区块头中的Merkle根哈希相匹配。如果全节点欺骗我们，并生成了伪造的交易数据，那么其他节点在验证过程中将会发现Merkle根哈希不匹配，从而拒绝接受这个区块。

1. **实验总结**

学习了基本区块结构及其必要属性，掌握了Merkle树的基本原理，学会计算区块的哈希值，实现了基于POW共识的挖矿过程 ，实现了账户类与钱包功能，实现了UTXO类与解锁脚本功能，实现了账户签名交易，矿工验签功能，学习了SPV 轻节点模块与验证。还体会了新旧两种代码结构的区别。为以后的学习打下了基础。