实验二 签名、验签以及UTXO的简单实现

【实验介绍】

在上个实验项目的基础上添加签名、验签和UTXO功能模块。

区块链的安全性由密码学保障：

私钥签名和公钥验签保证了交易的防篡改性，

UTXO的解锁脚本保证只有账户本人才能使用该UTXO。

UTXO结构节省了区块链的存储空间，区块链不必存储每个账户的余额，查询余额时仅需遍历所有的区块，计算该账户可使用的UTXO的总额，这种计算量相对于找随机数的可忽略不计。

本次实验将在旧有代码上添加账户（拥有公私钥）和UTXO模块，并修改其他模块以适配这种扩充。

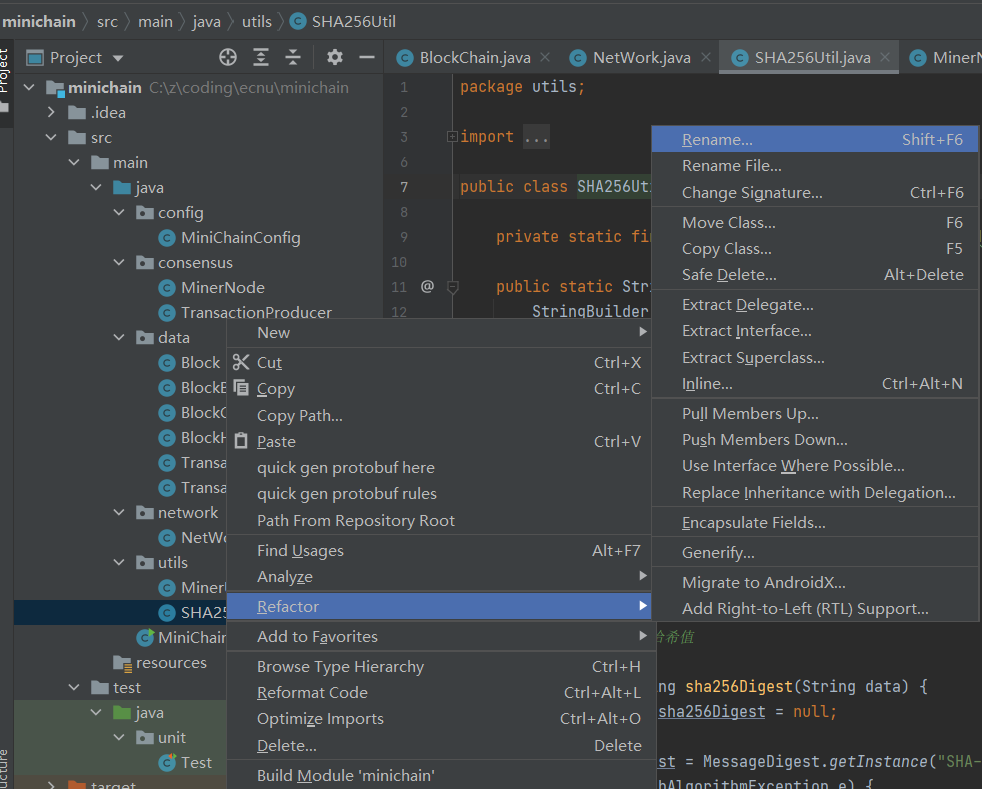
【实验要求】

1. 实现账户类的结构和计算钱包地址
2. 实现UTXO类的结构和解锁脚本
3. 实现账户签名交易，矿工验签检查功能

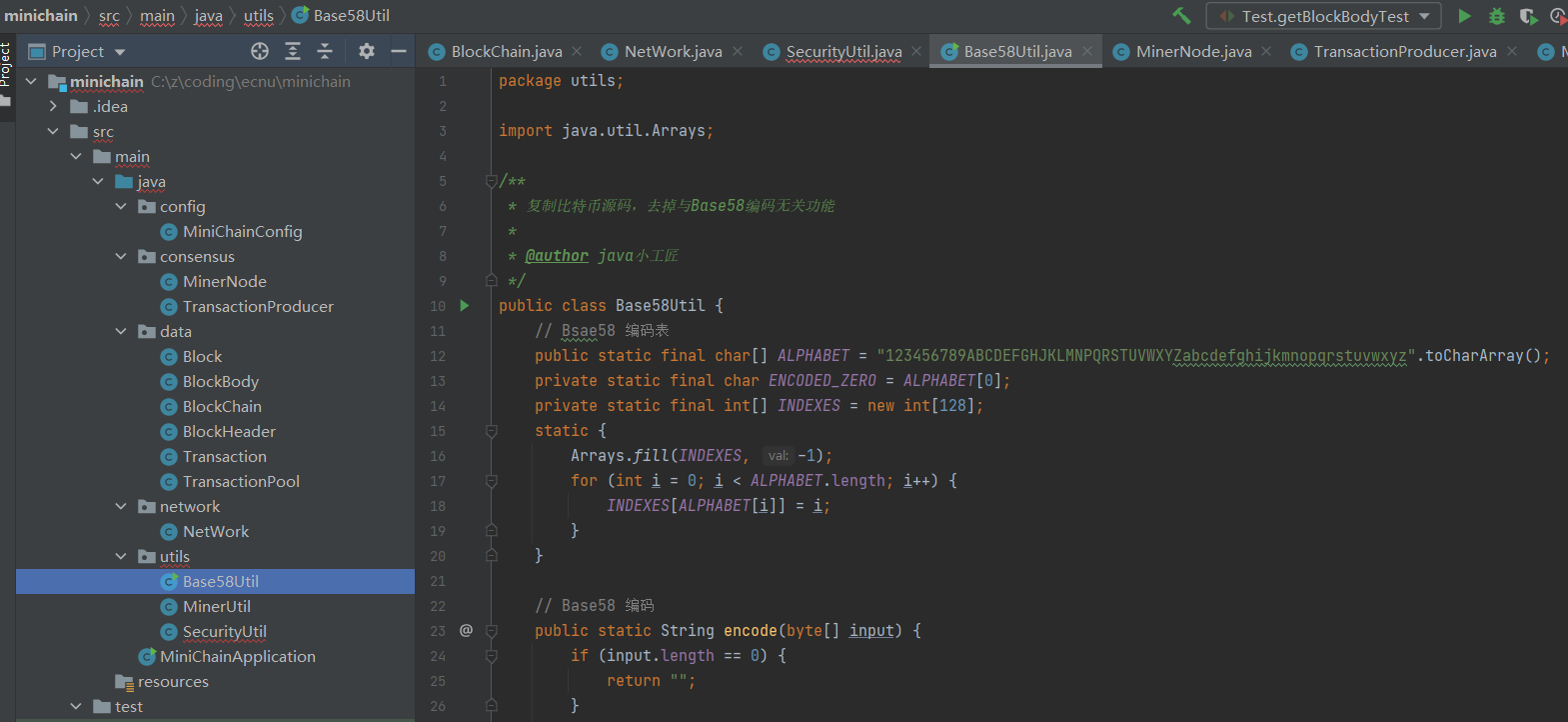
【实验过程】

1. Util工具类支持

右击SHA256Util，重命名文件为SecurityUtil，然后将提供的SecurityUtil中的内容复制过来（覆盖原有代码）。

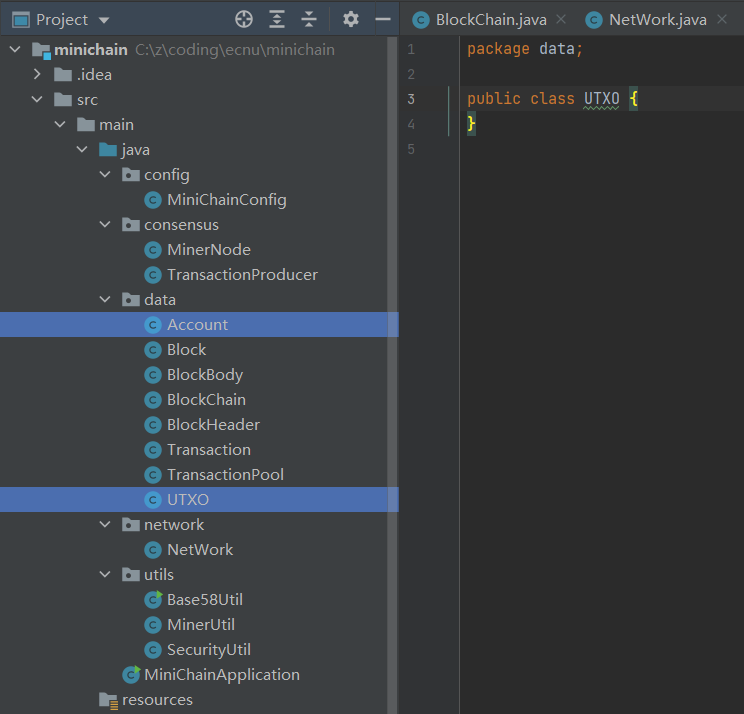


在utils包下新建类Base58Util，将提供的Base58Util中的代码复制过来。

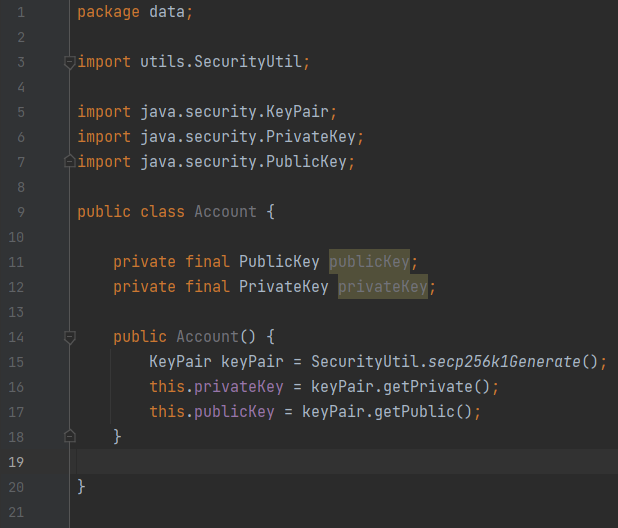


1. Account类

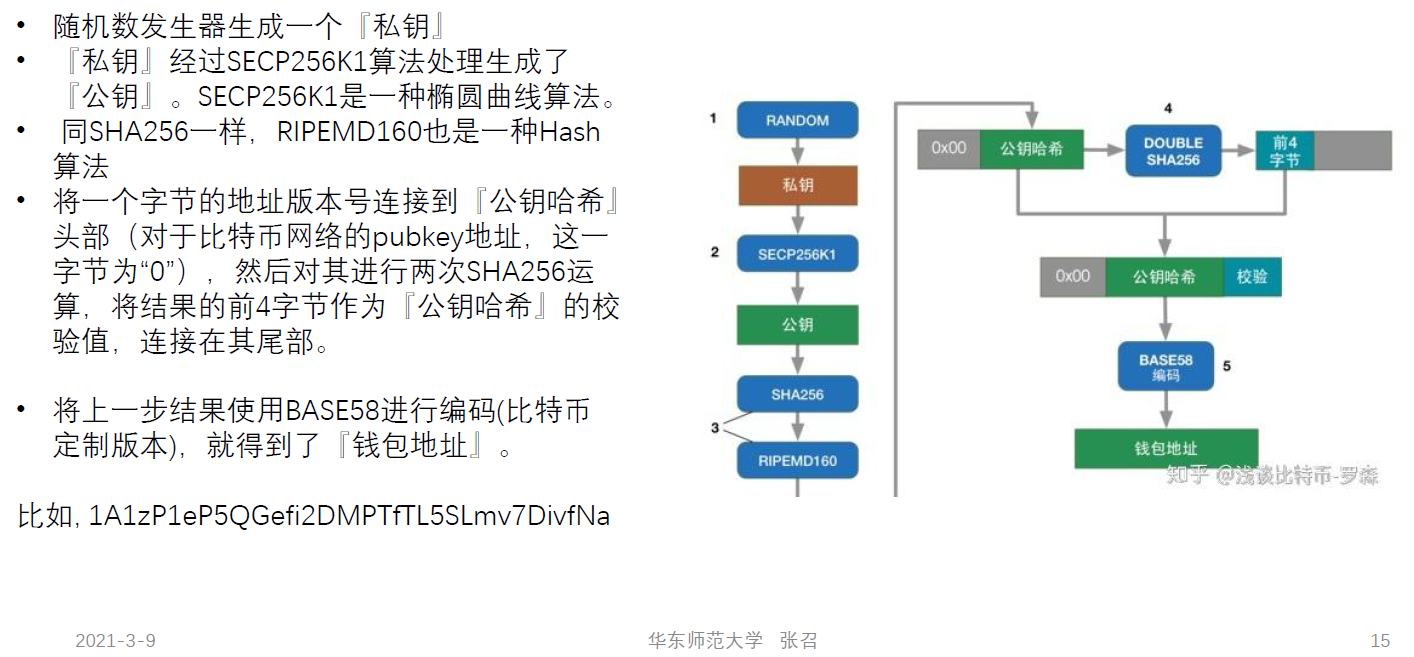
在data包下新建Account类和UTXO类。



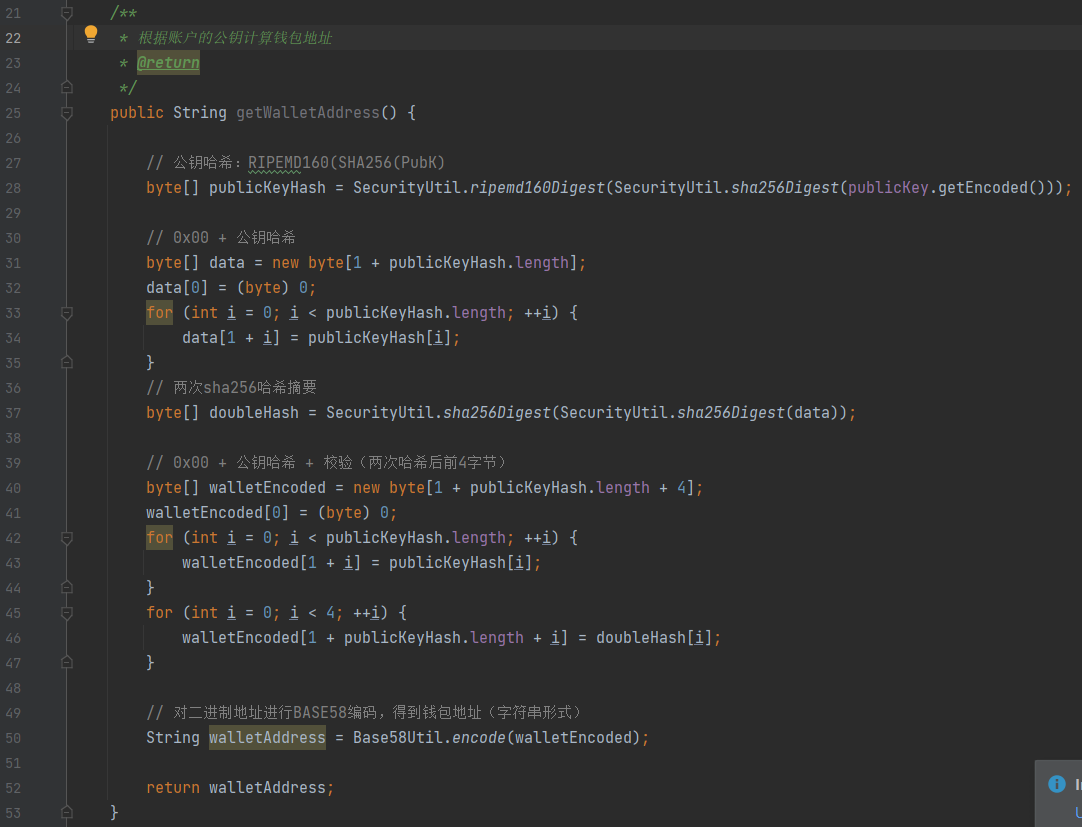
账户拥有公私钥，其通过椭圆曲线加密算法获得，故写下如下代码：



账户的私钥用来签名，公钥用来验签和获得钱包地址。首先我们编写生成钱包地址的代码，根据理论课上的知识，钱包地址生成算法如下：



故我们相应的编写对应的代码实现获得钱包地址的函数：



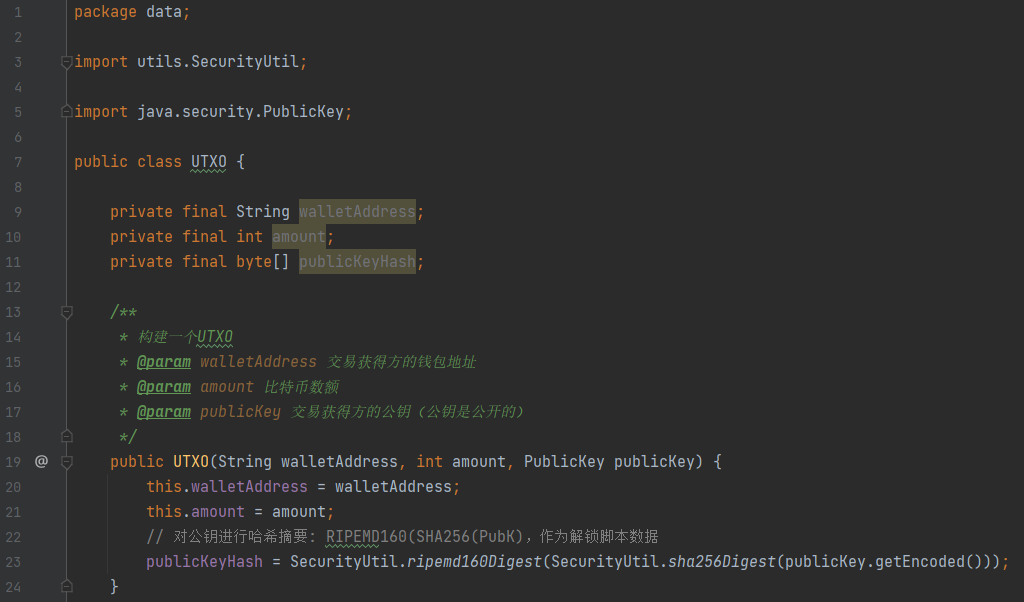
按下Alt+Insert快捷键快速生成相应的getter函数和toString函数，修改toString的内容让其公私钥输出十六进制字符串形式：



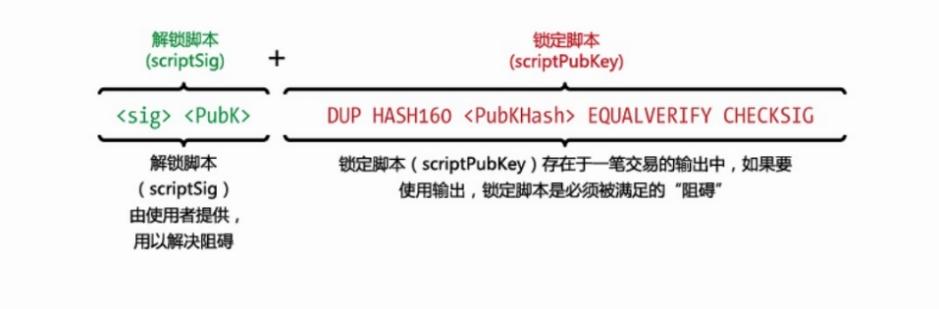
1. UTXO类

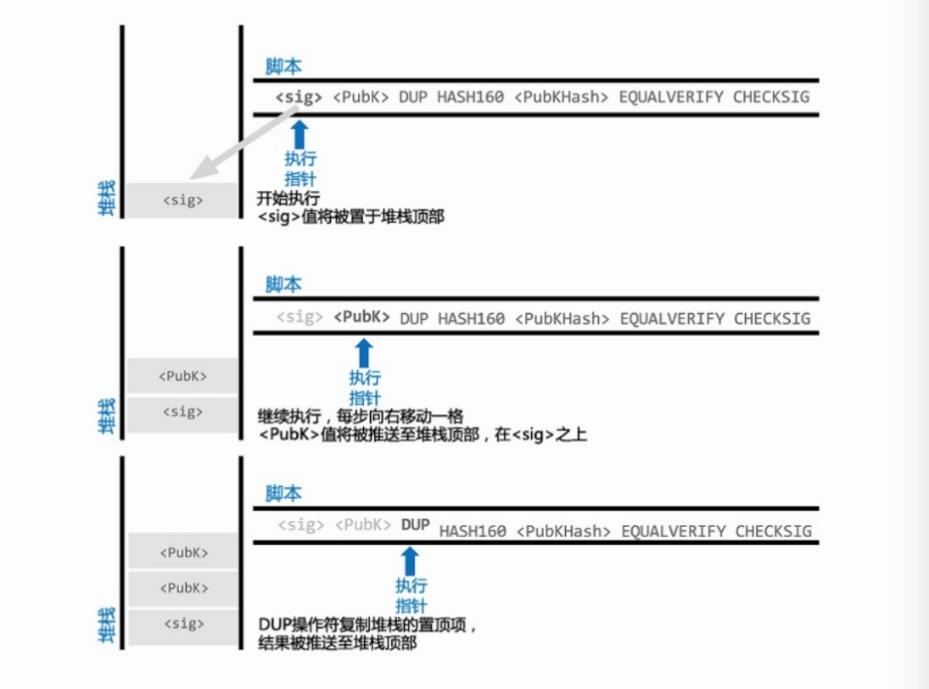
上一步我们已经新建了UTXO类，对于UTXO，它相当于账本上记得一笔账的输出，即“谁得到了多少钱”，归属于某个账户，可用来支付给别人。故UTXO需拥有的数据成员有：（1）“谁”，即账户钱包地址walletAddress（2）“多少钱”，即得到的金额amount。

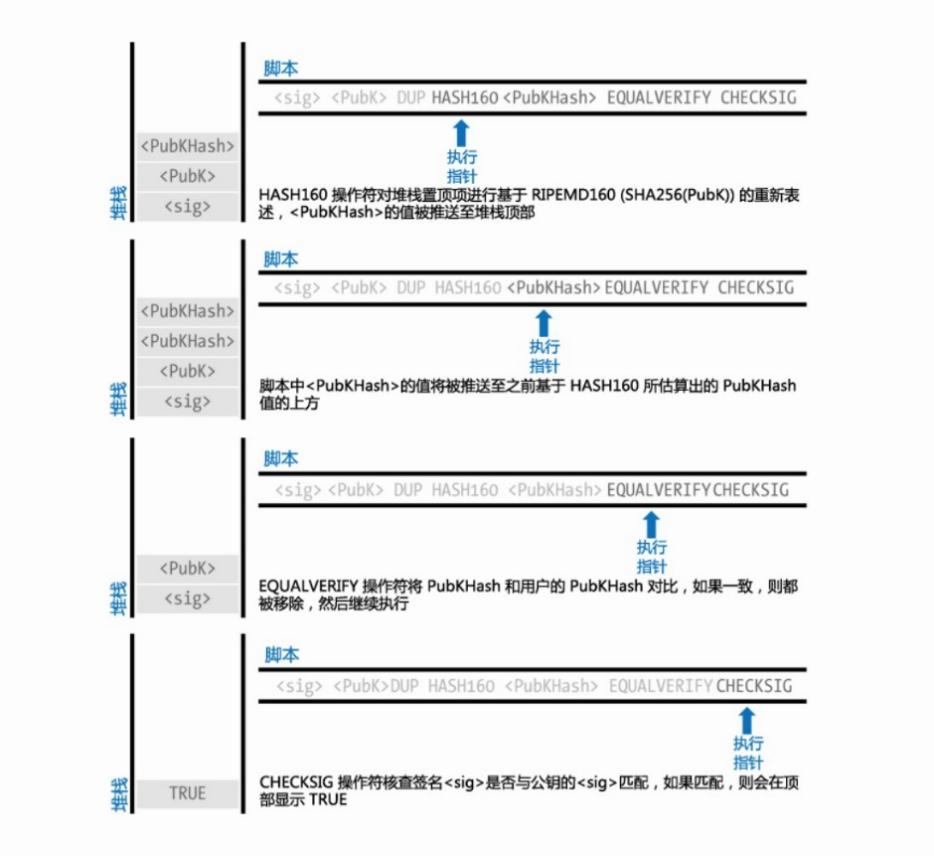
当然，这个UTXO只有该账户才能使用，需要该账户提供私钥的签名进行相关解锁脚本的操作，故该UTXO还需保存该账户公钥的相关信息，根据BTC的设计，这里是公钥的哈希值RIPEMD160(SHA256(PubK) 。所以，还有一个数据成员publicKeyHash：



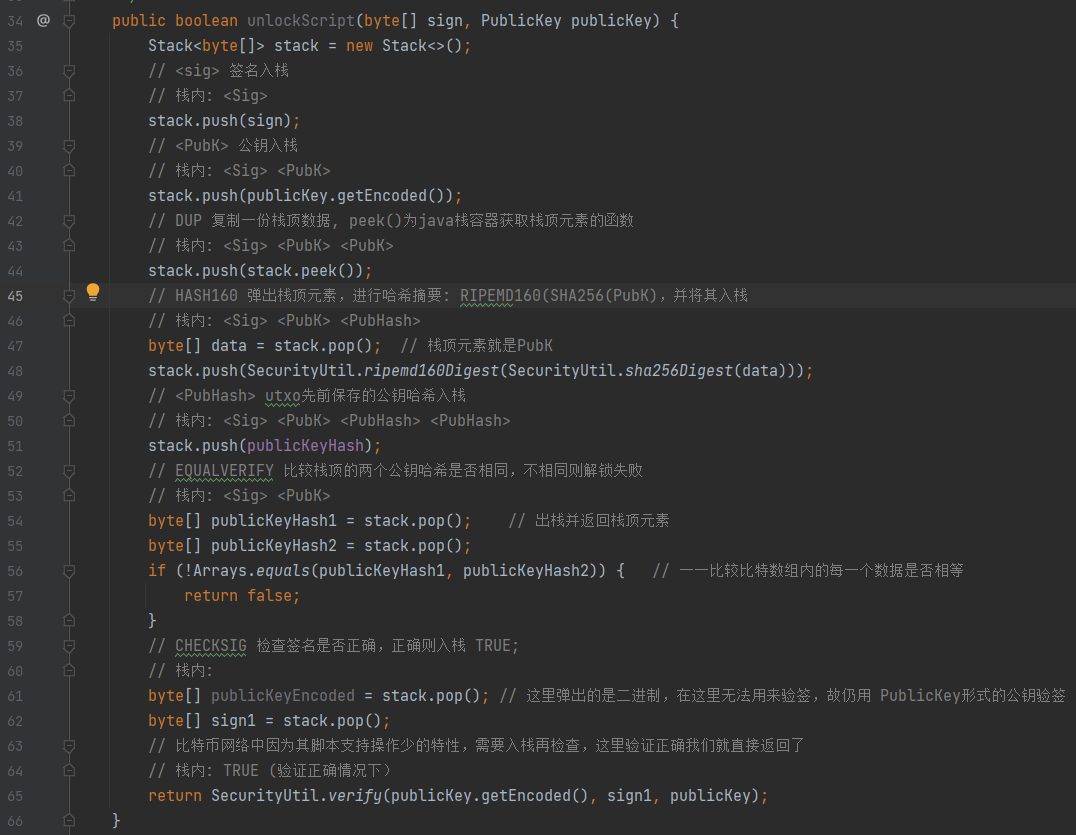
上面我们提到了UTXO需要解锁才能使用（防止被他人使用），比特币支持一种基于栈的逆波兰表达式的编程脚本语言（以太坊就支持更高级更抽象的智能合约了），相关解锁流程如下:



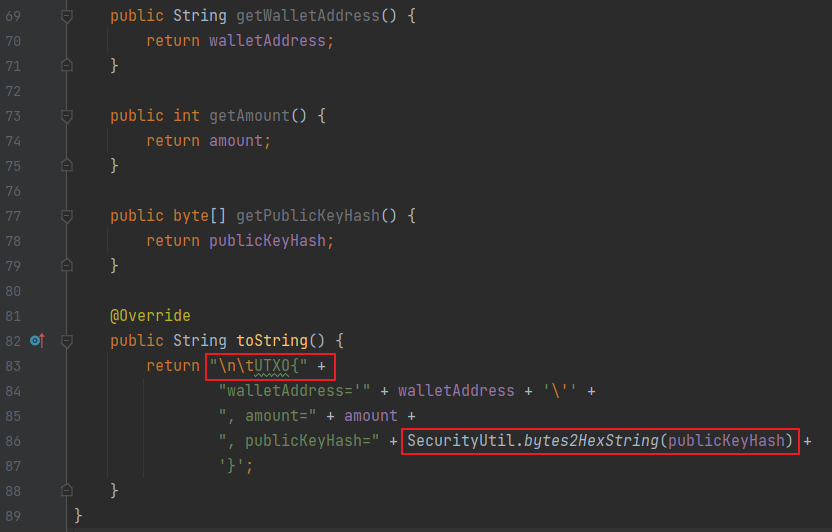




基于此我们可以模拟出相关的流程（java语言相对比较高级），构建一个解锁脚本函数：



Alt+Insert快捷键快速生成相应的getter函数和toString函数，修改toString的内容让公钥哈希输出十六进制字符串形式，添加换行符和制表符美化输出：



有了UTXO后，我们就可以计算某个账户的余额，就是将未使用的UTXO汇总，回到Account类中，添加一个计算账户金额的函数：

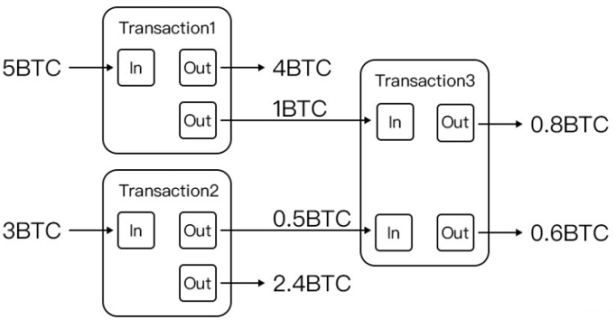


该函数的参数是trueUtxos，因为有些UTXO已经被使用了，但我们无法给它们换个类名，姑且将其算作fakeUtxos，相对而言还未使用的UTXO的集合就是trueUtxos。

至此，我们完成了新类的添加，下面将在其他模块内进行修改。

1. Transactions类

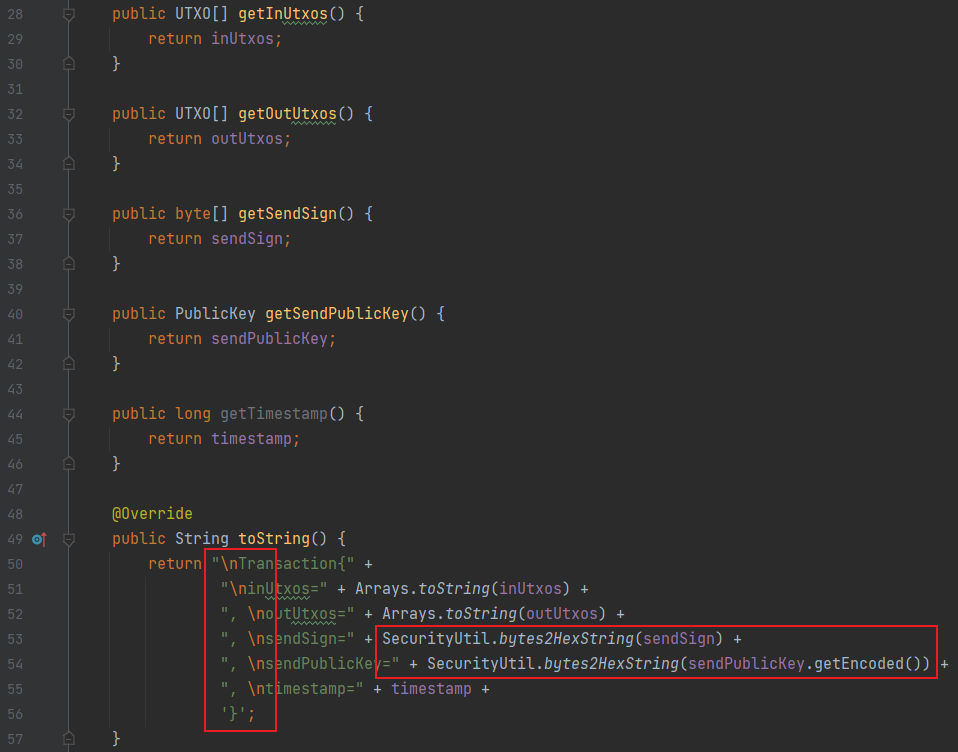
之前我们的交易类的数据就是一个名为data的字符串（String），现在我们实现了UTXO，所以交易中要存放UTXO，回顾理论课的内容，如下图所示，一个交易包含若干个输入UTXO和若干个输出UTXO：



故删除旧有的内部代码，修改Transaction类结构如下：

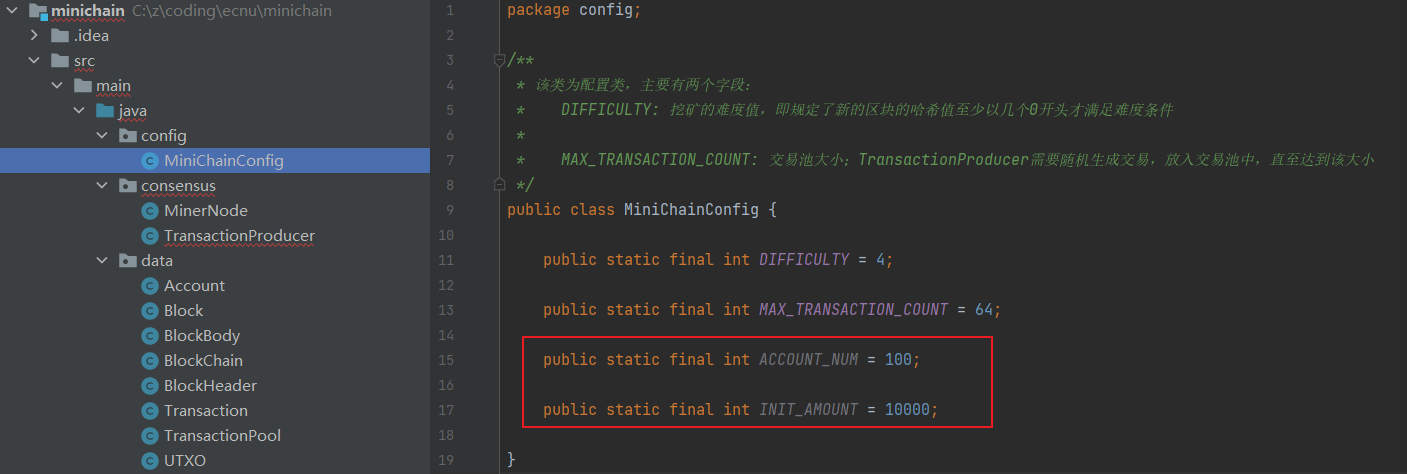


Alt+Insert快捷键快速生成相应的getter函数和toString函数，修改toString的内容让签名公钥输出十六进制字符串形式，添加换行符美化输出：



1. 添加账户

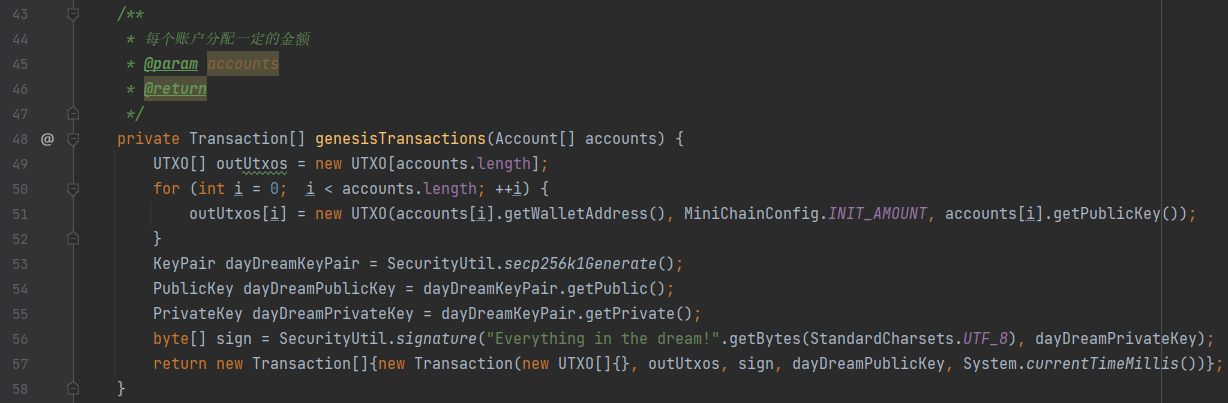
我们将初始化一些账户，并为他们提供一个固定金额的UTXO，由于已有结构的限制，该账户数组将作为BlockChain类的数据成员，先在MiniChainConfig配置类中配置相关参数，初始化账户为100个，初始UTXO金额为10000：



随后在BlockChain类中添加Account账户数组，然后修改构造函数，在初始区块中添加为这些账户提供的UTXO，这样每个账户在开始时就拥有10000比特币（以当前的时间点算是10000x60000$，人人都是亿万富翁:)）。



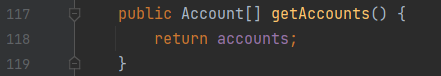
当然，为了让这些人凭空成为亿万富翁，我们得实现genesisTransactions函数，在当前类中添加相应函数，该函数会创建一批输出utxo，为每个账户提供一笔金额，只要该账户提供它的身份证明解锁便可使用。至于是谁平白无故地支付这么一笔交易并签名，代码中已经给了答案:)：



在当前类中，我们还需要实现一个函数，以便构造后续交易使用，那就是查询出某个账户可使用的UTXO（有些已经被使用了），在签名的计算账户余额中，就需要这么一个trueUtxos数组作为参数。而查询某账户可使用的trueUtxos需要遍历整个区块链每个块中每个交易的inUtxos和outUtxos，并判断它们的钱包地址是否是符合当前查询的地址。（代码中使用到Java的Set，这是哈希表，相当于C++中的unordered\_set和Python中的set，对于循环部分，可近似Python的循环来理解，C++也支持这种for循环。）

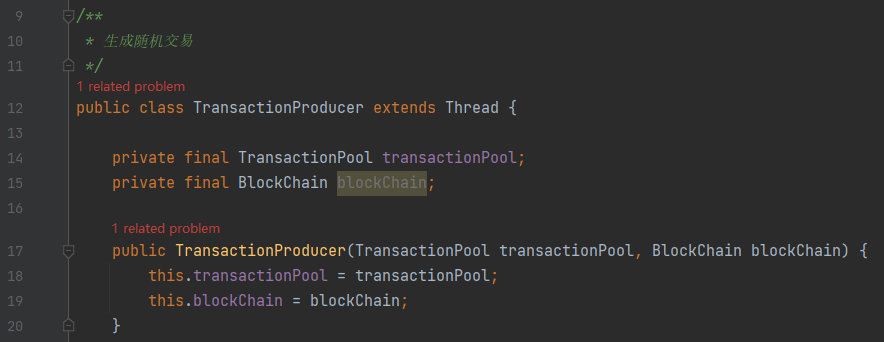


Alt+Insert快捷键快速生成相应的getter函数：



1. 随机生成一笔交易

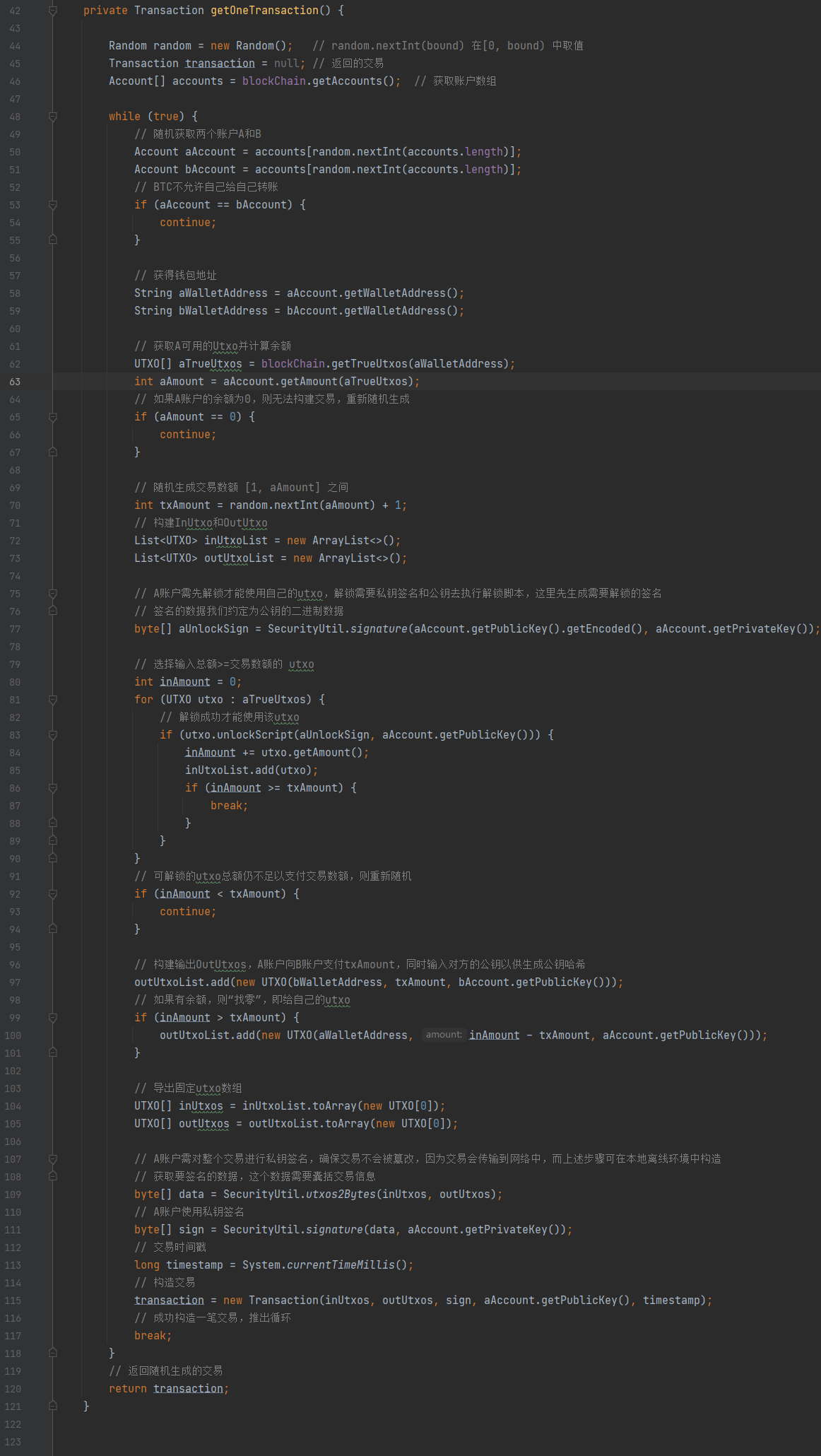
有了一群亿万富翁，也能查清他们的余额，他们就可以在链上开始交易了。之前我们的交易是随机数据，现在要开始真正的比特币交易。TransactionProducer类负责产生交易，现在它需要与账户和链的交互，故需要添加一个blockchain数据成员并修改其构造函数：



因为涉及到相关错误，点击上图红字跳转到NetWork类，修改相关代码：

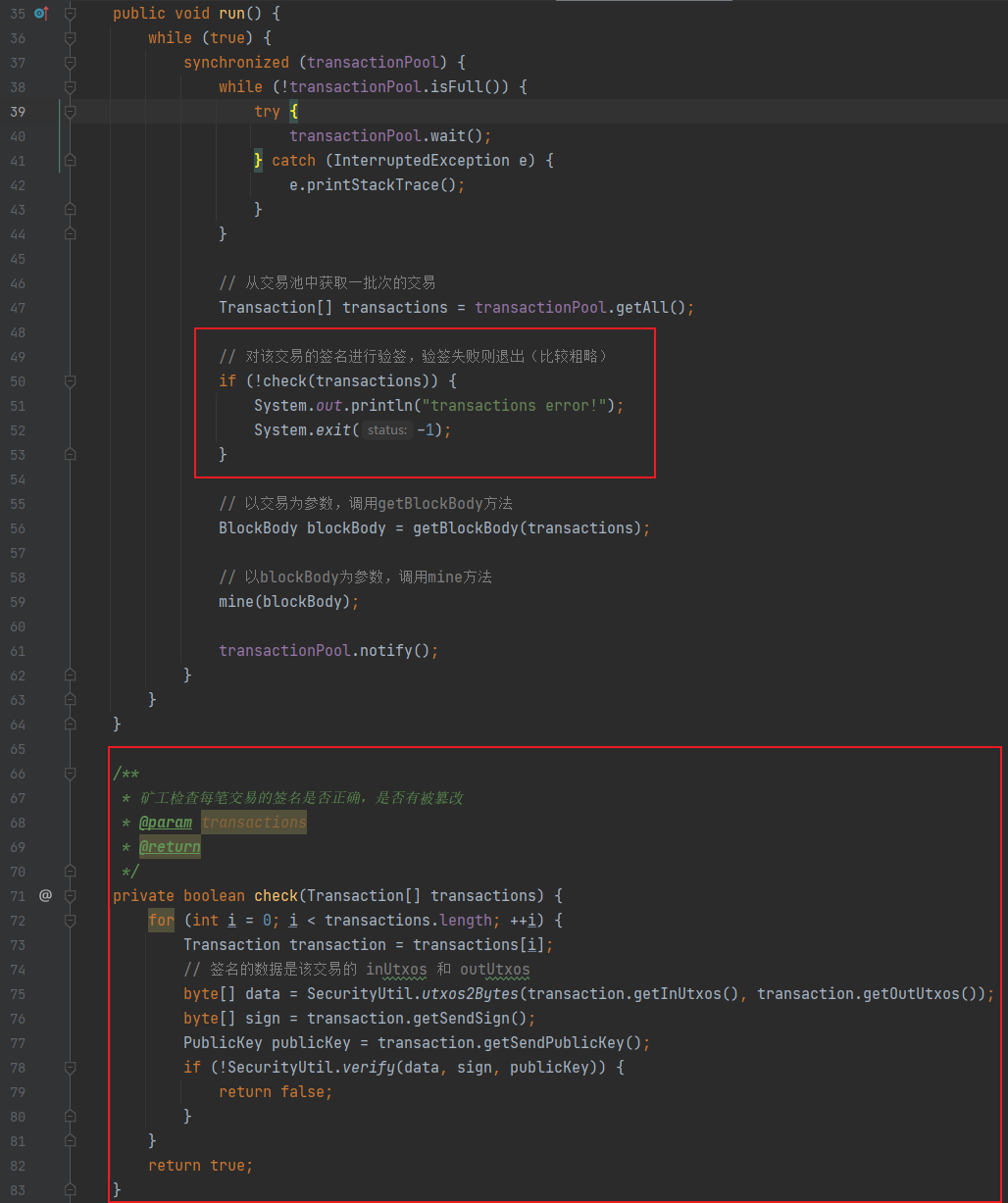


然后回到TransactionProduce类中修改getOneTransaction函数如下：



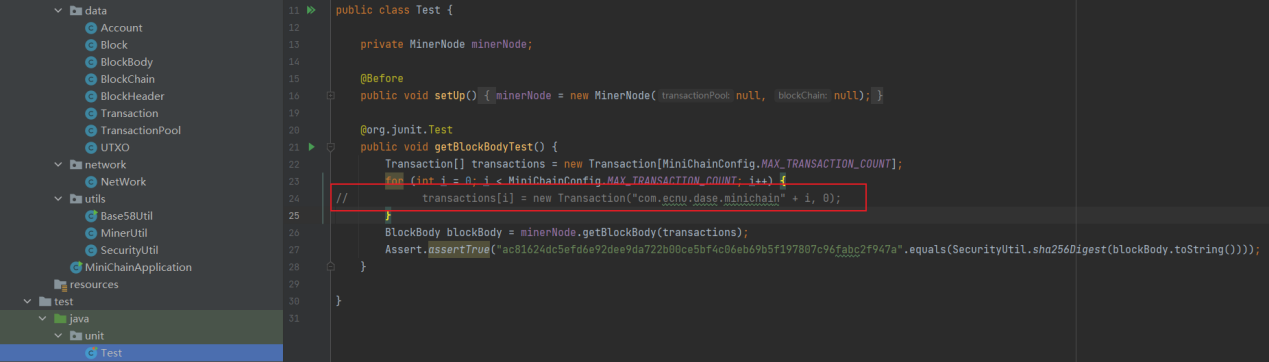
1. 矿工检查工作

因为交易得到了签名，所以矿工需使用构造交易的公钥进行验签，确保交易不被篡改，故MinerNode中添加如下代码（红框部分）：

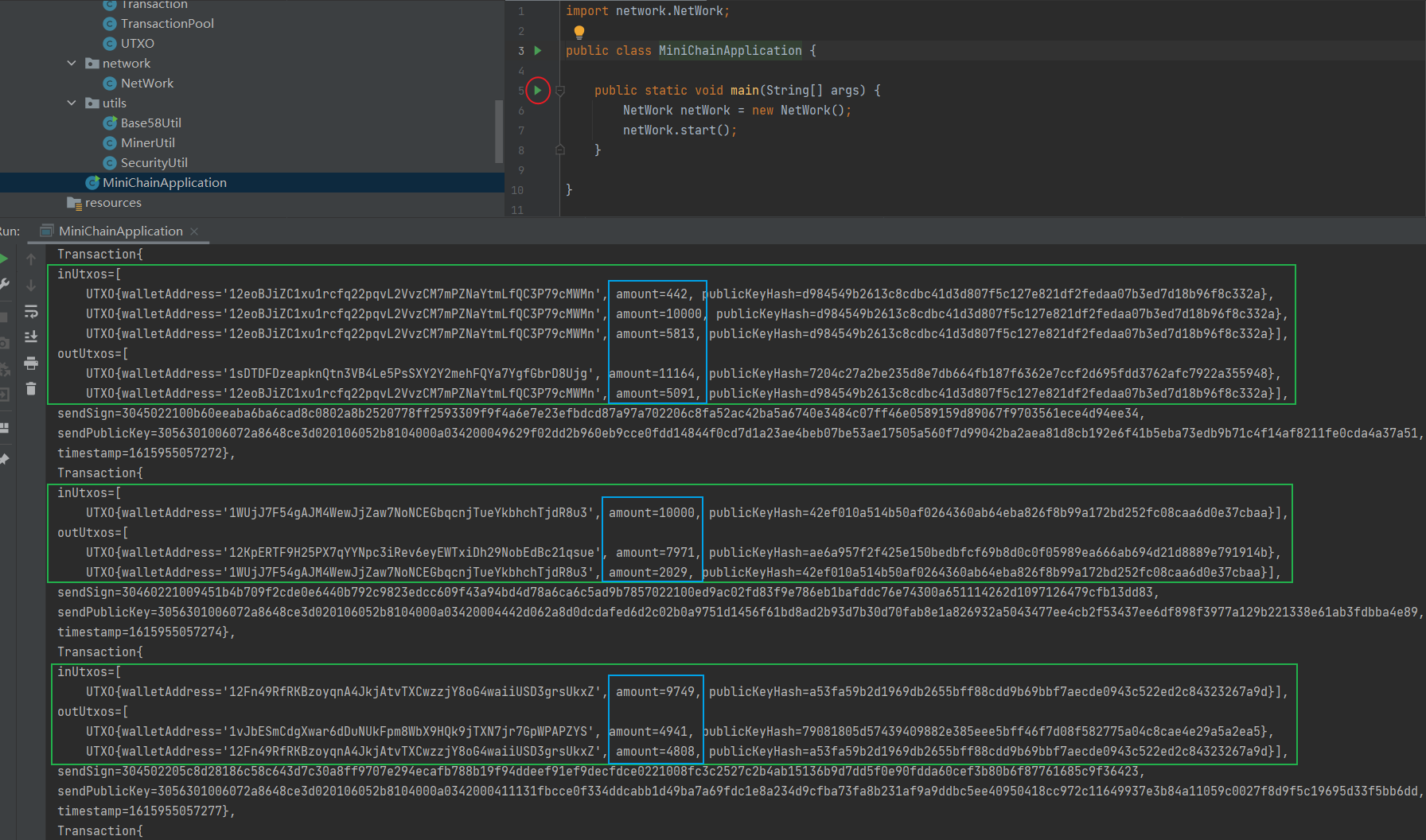


1. 运行

先将Test测试代码中因代码修改而出错的地方注释：

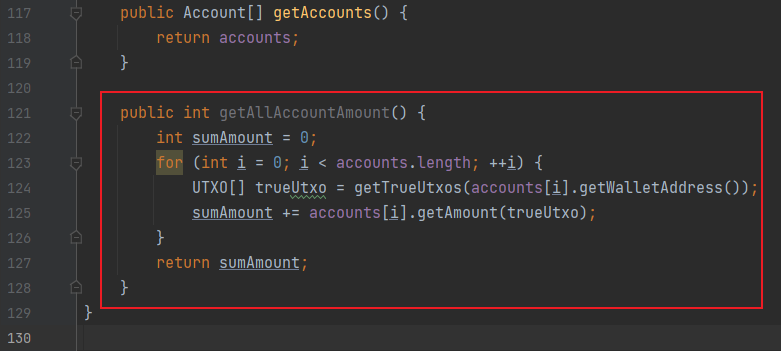


然后运行main函数启动minichain，可在打印信息中看到每个新块的信息（如果很长时间无打印信息可将MiniChainConfig里的DIFFICULTY参数降低），里面有若干个签名交易，交易中有若干输入UTXO和输出UXTO。

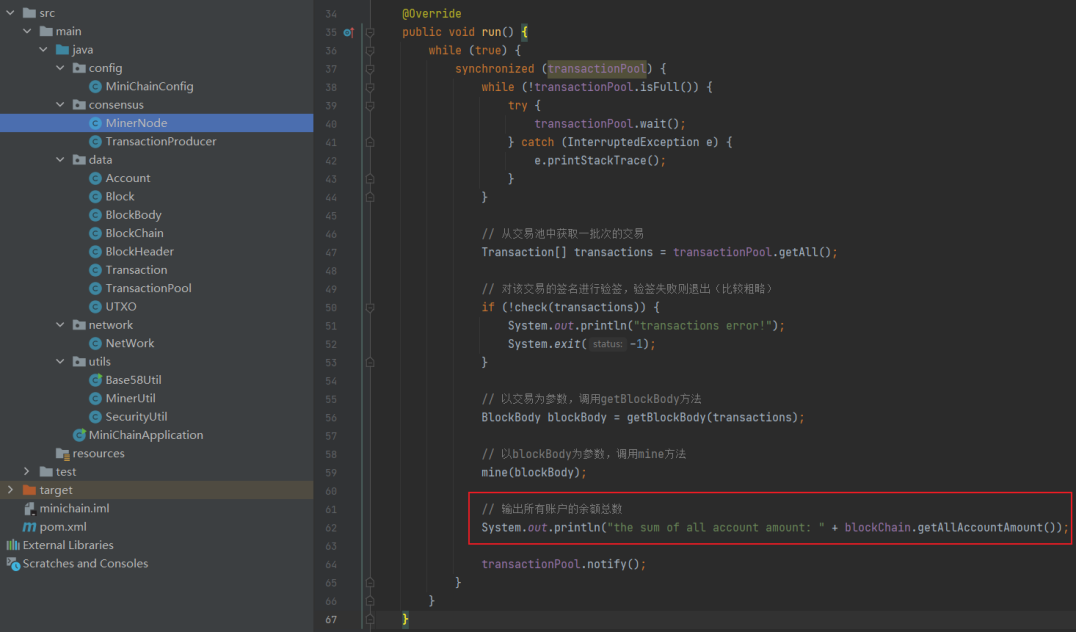


1. 验证

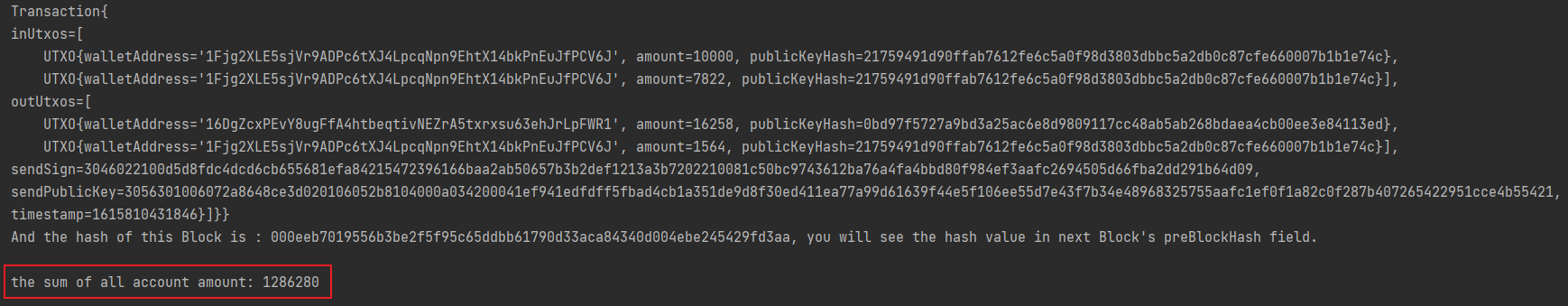
为了验证交易中的UTXO是否正确进行中，我们可以每出一个块就统计下所有账户的总余额，如果一直保证在账户数x账户初始金额（这里是100000），就能在一定程度上保证程序运行正确。故先在BlockChain类中添加统计下所有账户的总余额的函数：



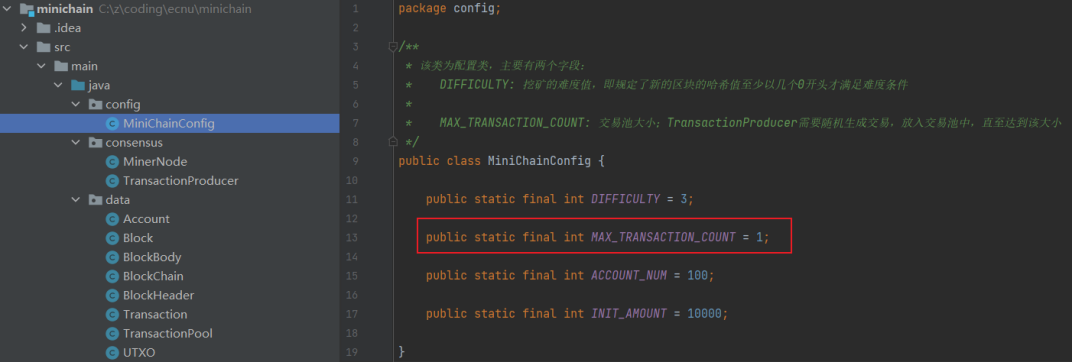
然后在MineNode类中添加输出代码：



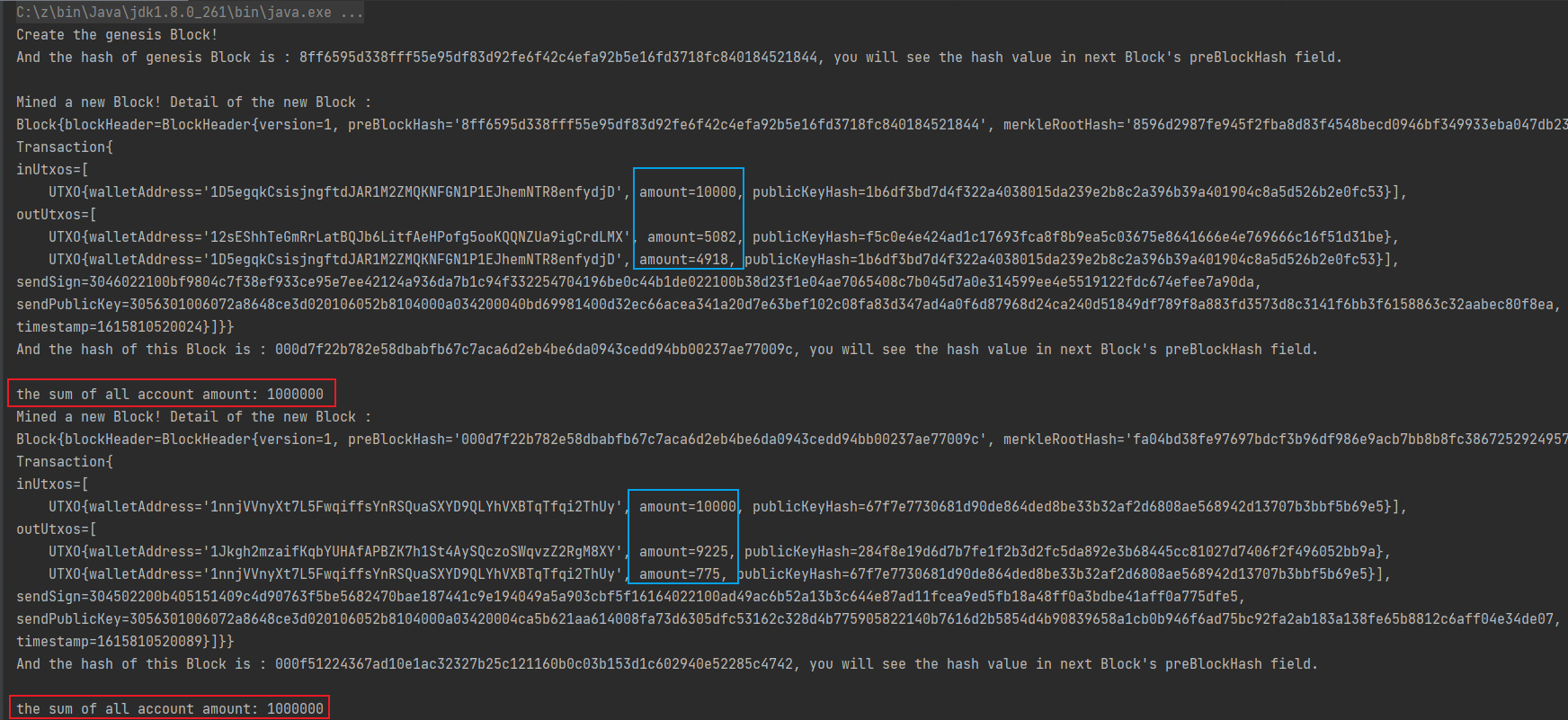
然后运行程序：



发现总金额不一样……考虑原因，是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，故会让总金额慢慢变大，在实际过程中，会有更多更严格的检查机制，我们这里是随机生成的。虽然可以用相关容器做修改，但会带来更多冗余的代码量，所以，我们适当做些妥协，将每个块的交易数限制为1个（默克尔树计算不受影响）：

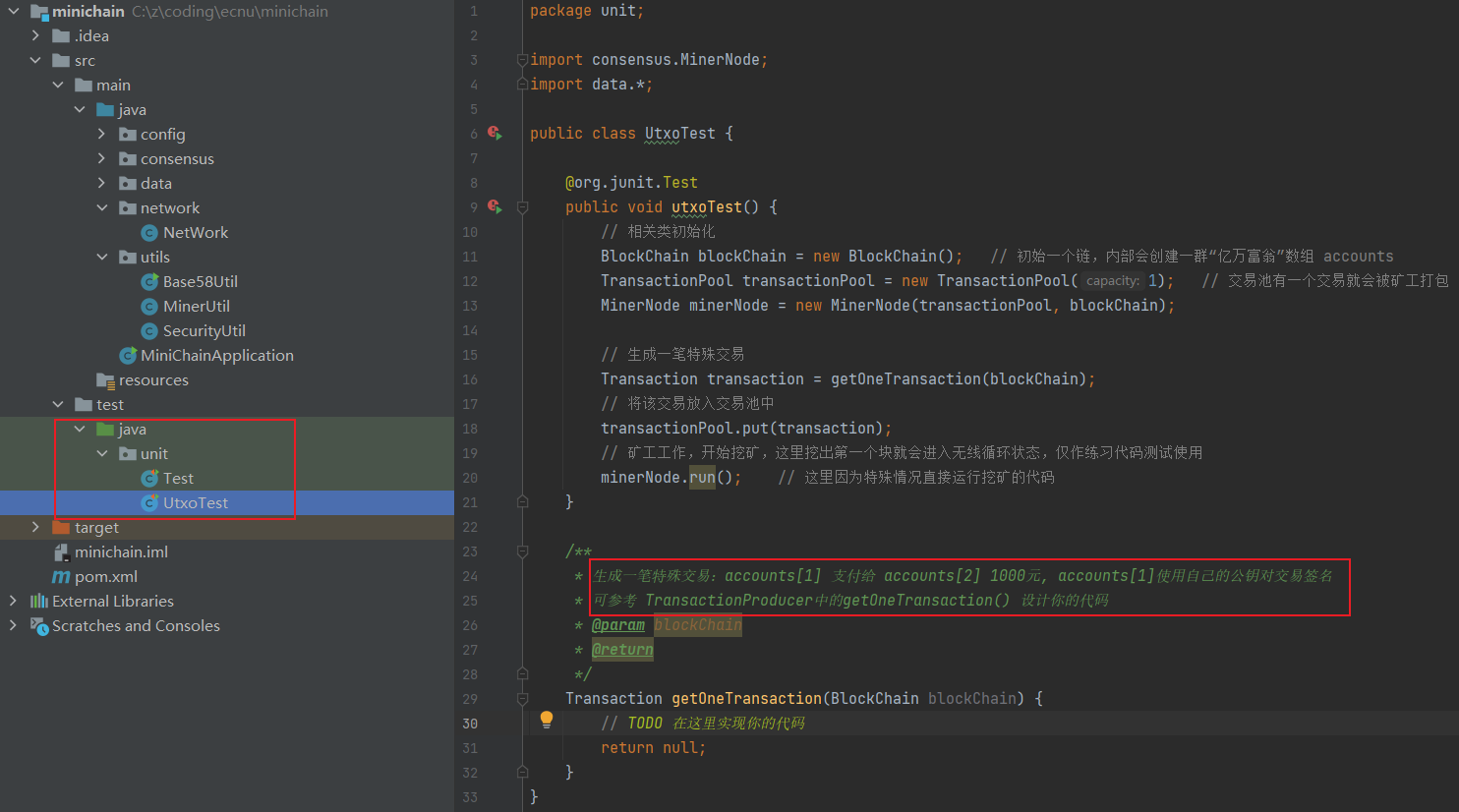


然后运行程序，总金额不变，且一个交易中输入UTXO和输出UTXO金额相等。

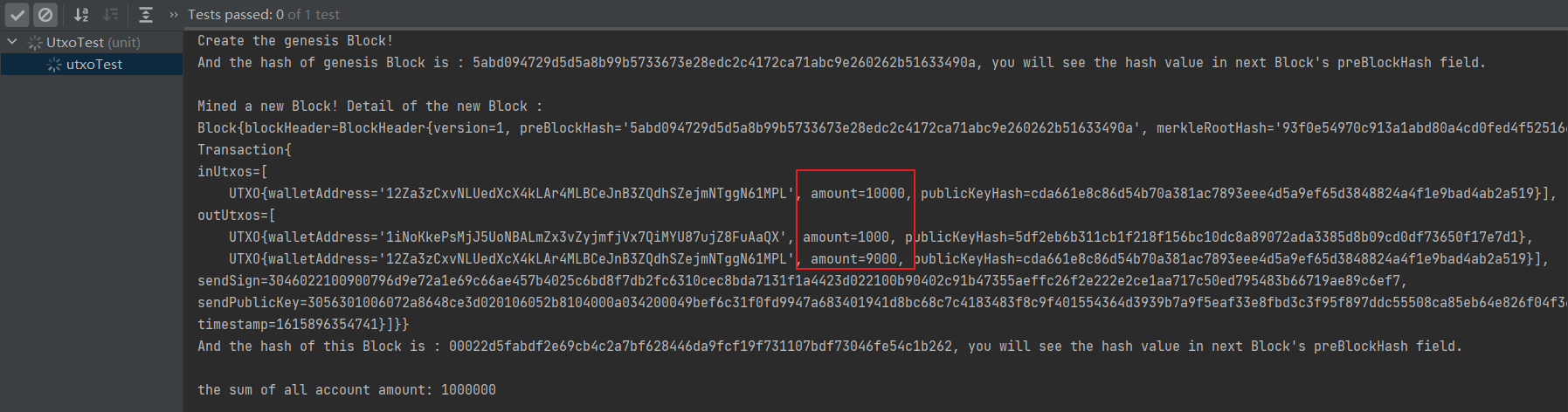


1. 练习题
2. 构造一个确定的UTXO交易

在一个新链中，这些亿万富翁开始第一笔交易：accounts[1]支付给accunts[2]1000元，然后签名交易放入到交易池中，区块参数设定为1个交易就打包成块，故这笔交易立刻将被上传至链上。在下图位置创建UtxoTest类，填补相关函数，实现该笔交易的构造代码：



结果如下（地址和公私钥每次都会不同）：



1. 构造一个复杂的UTXO交易（选做）

在上述交易中，我们只要遍历区块链的所有UTXO，就很容易就知道谁给谁转账，隐私无法得到保障，一种提供隐私保护方案是将多方交易混合。当前有交易txa：账户1给账户2转账1000元，有交易txb：账户3给账户4转账1000元，这时可将这两笔交易合并为一笔交易，由四方共同约定产生的私钥来签名（这里可使用相关api生成）。参考练习题（1）代码实现上述相关的交易。