**实验2 签名、验签以及UTXO的简单实现**

**【实验介绍】**

在实验1中，为了方便的建立起一个区块链系统，我们简化了交易的结构，仅仅使用UUID作为交易的内容。以比特币为例，交易的内容一般为用户转账，携带着用户的信息和转账金额数。为了保障用户在网络中的匿名性和交易的可验证性和安全性，区块链系统一般使用地址来指代某个用户，并使用数字签名（Digital Signature）技术来辨别交易的真伪。同时，为了解决数字货币中的双花问题，即一笔钱被花费了两次，比特币使用UTXO（Unspent TX Output）作为交易模型。本次实验的内容是在实验1的基础上添加账户模块（可看作一个简单的钱包）和UTXO模块，并实现交易的签名及验签功能，同时会修改其它模块以适配新功能。

在进行实验操作之前，我们首先了解非对称加密和数字签名的相关概念以及学习UTXO模型的工作原理。

1. 非对称加密

非对称加密是指在使用前生成一个公钥(Public Key)和对应的私钥(Private Key)，在加密明文和解密密文的过程中，使用不同的密钥，即一个用于加密，一个用于解密。非对称加密虽然速度较慢，但其公私钥分开的优点，不用向外分发解密的私钥，安全性大大提高，多应用于签名验证和数字身份场景。

在非对称加密算法应用中，用私钥加密的数据要用对应的公钥才能解开，用公钥加密的数据要用对应的私钥才能解开。常见的非对称加密算法有RSA、DSA（Digital Signature Algorithm）数字签名算法、ECC（Elliptic curve cryptography）椭圆曲线密码算法、ECDSA（Elliptic Curve Digital Signature Algorithm）椭圆曲线数字签名算法——DSA和ECC的结合。

比特币中的私钥本质是一个密码学安全的随机数，通过secp256k1（一组特定参数的ECDSA）变换后得到公钥，公钥再经过SHA256和RIPEMD160哈希(两者结合又称Hash160)，添加前导字节并进行Base58Check编码，得到可读性高的短地址。从私钥到公钥再到地址都是单向的算法，以目前的算力几乎是不可逆的，保证了用户私钥的安全性。比特币中私钥、公钥和地址的关系如图2-1所示。



图 2-1 私钥、公钥、地址关系图

1. 数字签名

数字签名在区块链中有着广泛的应用。从功能上来讲，数字签名与我们在纸质文件上的签名类似，是一种鉴别信息来源以及信息真伪的方法。从技术上来讲，数字签名使用密码学中的哈希（Hash）算法、非对称加密（Asymmetric Cryptography）算法来保证以下三点：①确认信息是由签名方发送的；②确认信息在传输过程中没有受到修改；③确认信息在传输过程中没有出现丢失。非对称加密实现数字签名如图2-2所示。

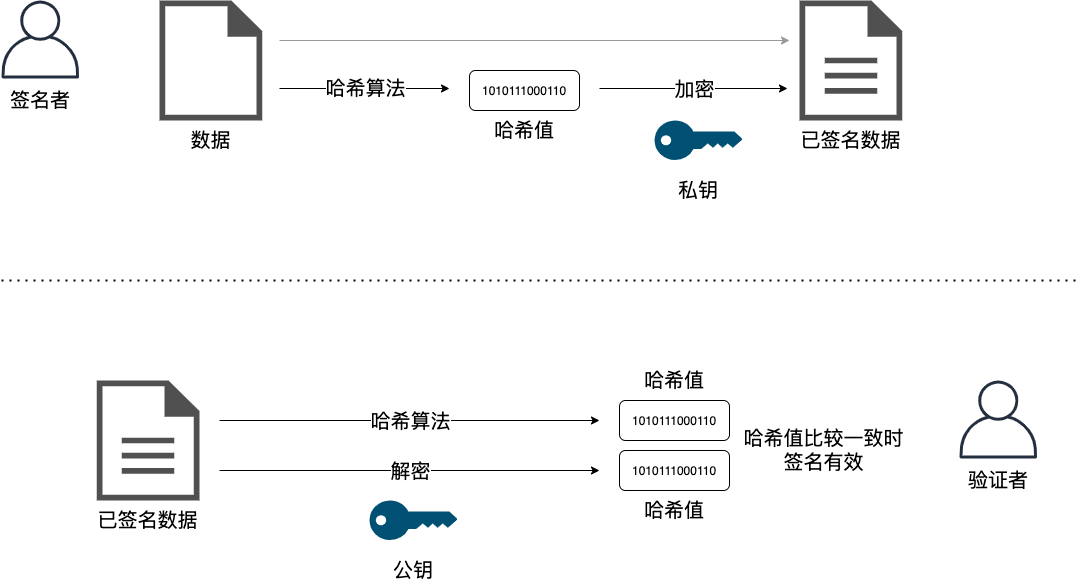


图 2-2非对称加密实现数字签名

1. UTXO模型

比特币的区块链由一个个区块串联构成，而每个区块又包含一个或多个交易。如果我们观察任何一个交易，它总是由若干个输入（Input）和若干个输出（Output）构成，一个Input指向的是前面区块的某个Output，只有Coinbase交易（铸币交易）没有输入，只有凭空输出。如图2-3所示，这些交易的Input和Output总是可以串联起来。



图 2-3 区块链交易间的关系

UTXO（Unspent TX Output）译名为未花费的交易输出，UTXO模型的设计基于一种思路：除了铸币（Coinbase）交易生成的比特币，任意一笔钱不会凭空产生，也不会凭空消失。UTXO由四部分组成；

1. address: 拥有此UTXO的地址；
2. amount: 此UTXO的金额；
3. Signature Script: UTXO解锁脚本，使用交易发送方私钥加密交易内容得到的签名；
4. Pubkey Script: UTXO锁定脚本，包含交易获得方的公钥哈希。

如图2-4所示，展示了Alice对Bob进行一笔转账交易的过程：假设Alice有 amount 个比特币，这其实意味着，之前有一个交易把这些比特币转入Alice的地址，这个交易的输出（即 amount个比特币）未被使用，Alice拥有了这 amount个比特币。

1. Alice使用公钥解锁自己的UTXO，构造新的UTXO和一笔转账交易；
2. Alice使用私钥对UTXO的交易内容进行签名；将UTXO的输出地址设为Bob的钱包地址，并把Bob的公钥Hash作为解锁此UTXO的凭据；
3. 矿工将这一交易打包进新的区块，转账交易完成。

经过6个区块确认（证明该笔交易极大概率合法）后，这 amount 个比特币就属于Bob了。实际上Bob拥有的是这笔转账交易的UTXO。



图2-4 基于UTXO模型，Alice转账给Bob

上面提及了比特币中的解锁脚本（Signature Script）与锁定脚本（PubKey Script）。比特币脚本是一种基于栈结构的无状态脚本语言，只能进行有限的操作，图灵不完备。虽然简单，但在数字货币领域，也意味着更少的金融风险。

在UTXO 模型中，交易只是代表了 UTXO 集合的变更。而账户和余额的概念是在 UTXO 集合上更高的抽象，账号和余额的概念只存在于钱包中。以比特币钱包为例，钱包管理的是一组私钥，对应的是一组公钥和地址。要查看钱包余额，必须从创世区块开始扫描每一笔交易，如果：

1. 遇到某笔交易的某个Output是钱包管理的地址之一，则钱包余额增加；
2. 遇到某笔交易的某个Input是钱包管理的地址之一，则钱包余额减少。

所以从狭义上来说，比特币钱包中并没有比特币，只有钱包地址关联的所有UTXO之和，代表着钱包地址拥有这些比特币的所有权。

**【实验要求】**

(1) 实现账户类与钱包功能

(2) 实现UTXO类与解锁脚本功能

(3) 实现账户签名交易，矿工验签功能

**【实验准备】**

**1. 环境配置**

本次实验使用和实验1相同的开发环境。

1. **导入代码**

推荐读者使用自己在实验1完成的代码上进行此实验。若仅想学习数字签名和比特币UTXO的工程原理，也可以直接导入附件里实验2的minichain压缩包进行实验，此压缩包含有实验1的参考代码。

**【实验过程】**

**1. Util工具类支持**

如图2-5所示，右击SHA256Util，重命名文件为SecurityUtil，然后将附件中实验2文件夹下的SecurityUtil.java中的内容复制并覆盖原有代码。

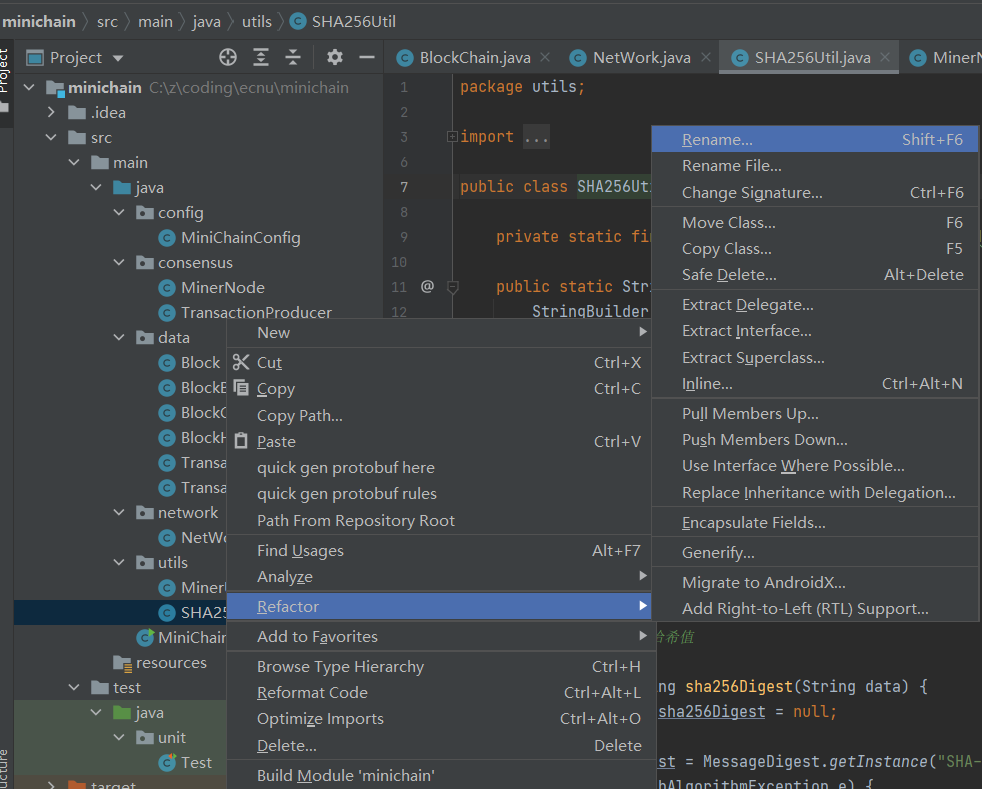


图2-5 重命名SHA256Util

在utils包下新建类Base58Util，将附件中实验2文件夹下的Base58Util.java中的内容复制并粘贴到此类下，结果如图2-6所示。

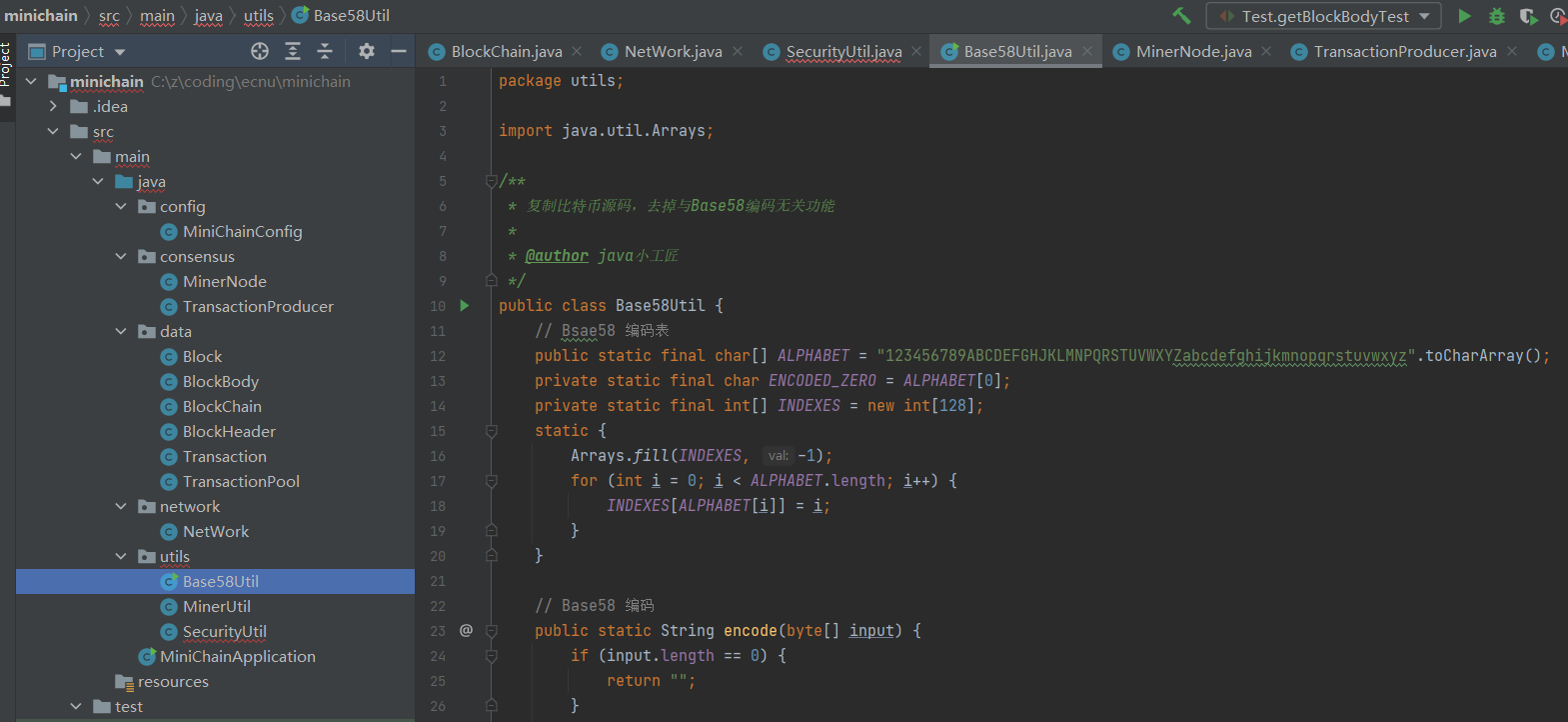


图2-6 Base58Util.java

1. **新建Account类、UTXO类**

在data包下新建Account类和UTXO类，如图2-7所示。

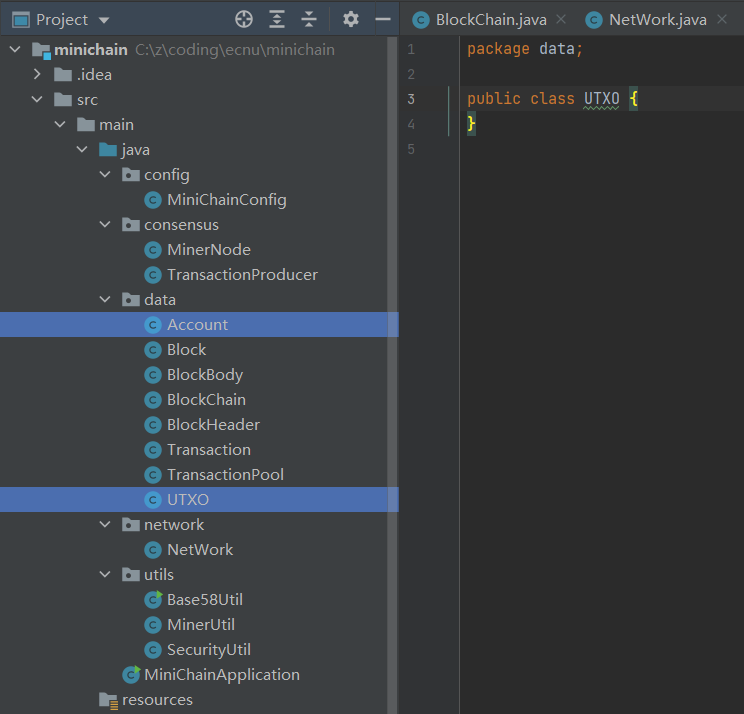


图2-7 新建Account类和UTXO 类

账户拥有公私钥，其通过椭圆曲线加密算法获得，故在Account类中添加构造方法如图2-8所示。

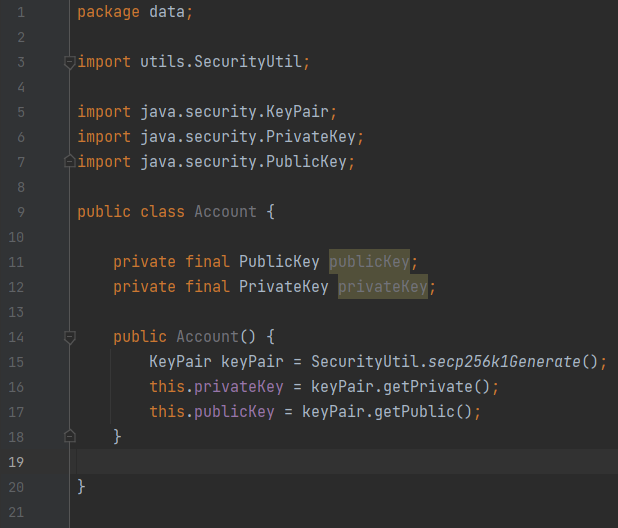


图2-8 Account构造方法

在实验介绍中，我们描述了比特币钱包地址的生成算法，算法流程如图2-9所示。



图2-9 比特币钱包地址生成算法

编写对应的代码实现获得钱包地址的方法，如图2-10所示。

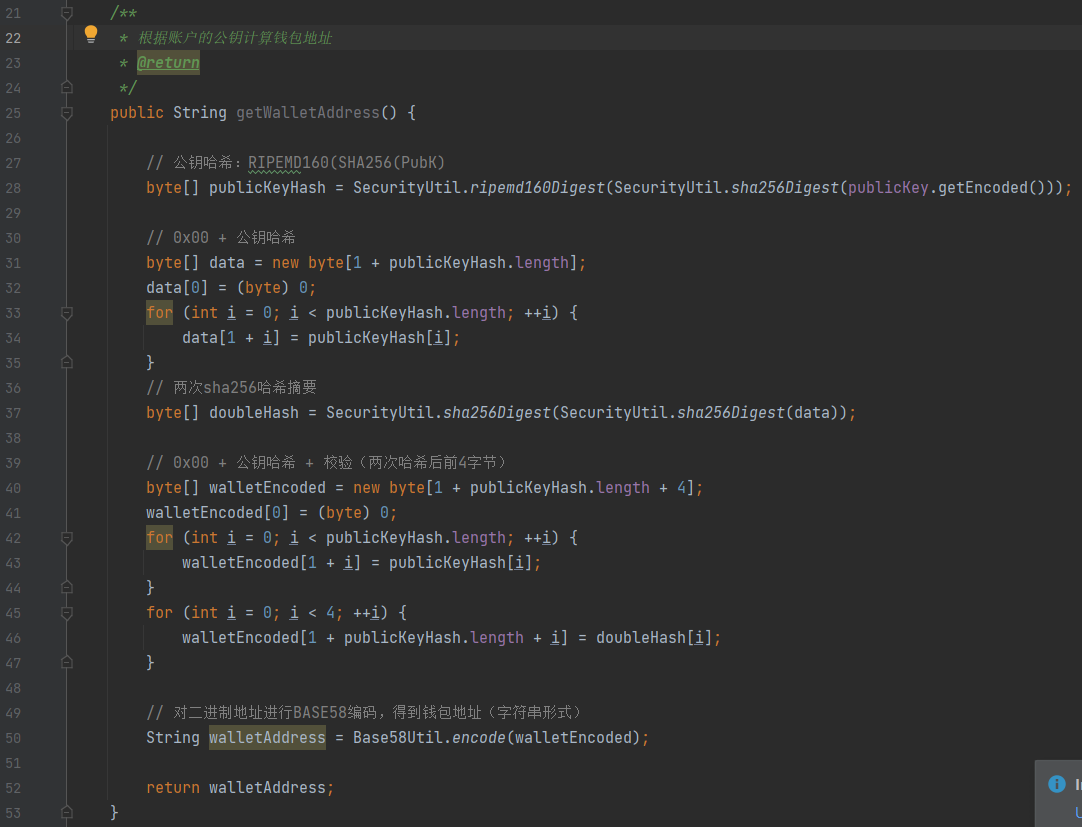


图2-10 Account类\_方法\_获取钱包地址

按下Alt+Insert快捷键快速生成相应的getter方法和toString方法，修改toString的内容使其公私钥的输出为十六进制字符串形式，代码如图2-11所示。



图2-11 Account类\_方法\_getter和toString

1. **完善Account类、UTXO类**

在实验介绍中，我们给出了UTXO具体的数据结构。我们可以忽略签名脚本的具体实现，完善UTXO类的构造方法。如图2-12所示。

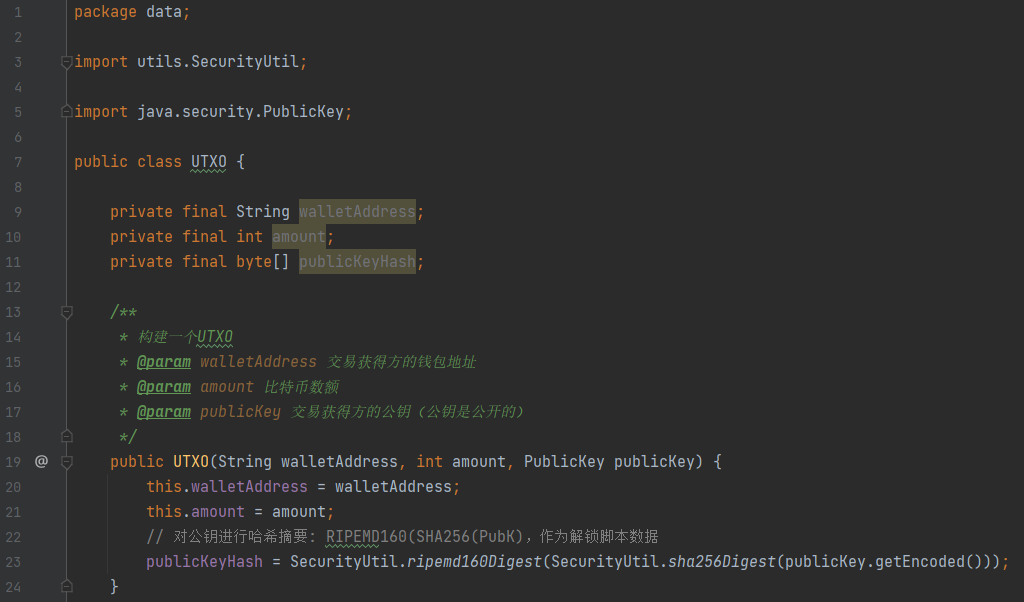


图 2-12 完善UTXO构造方法

实验介绍中，我们提到UTXO需要解锁才能被交易使用，解锁脚本与锁定脚本的模型如图2-13所示，解锁脚本算法如图2-14所示。



图2-13 解锁脚本与锁定脚本模型



图2-14 UTXO解锁脚本算法

基于上述算法，我们使用Java实现一个解锁脚本方法，如图2-15所示。

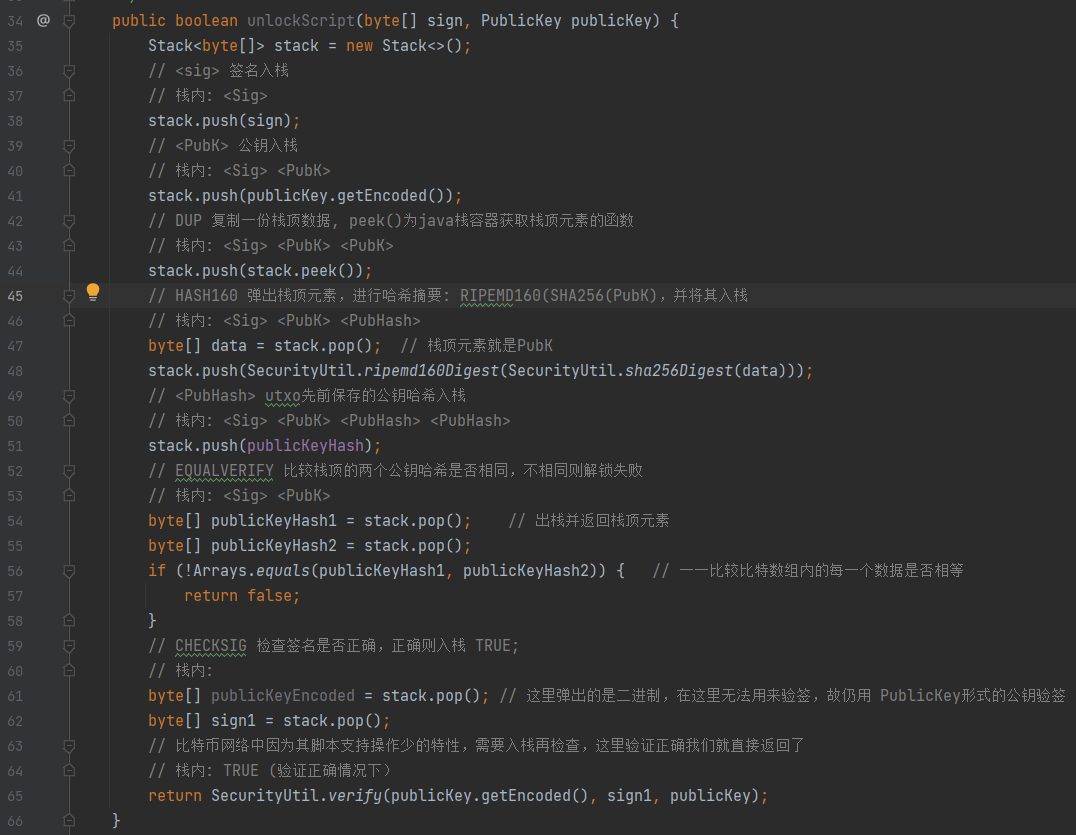


图2-15 UTXO类\_方法\_unlockScript

Alt+Insert快捷键快速生成相应的getter方法和toString方法，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，代码如图2-16所示。

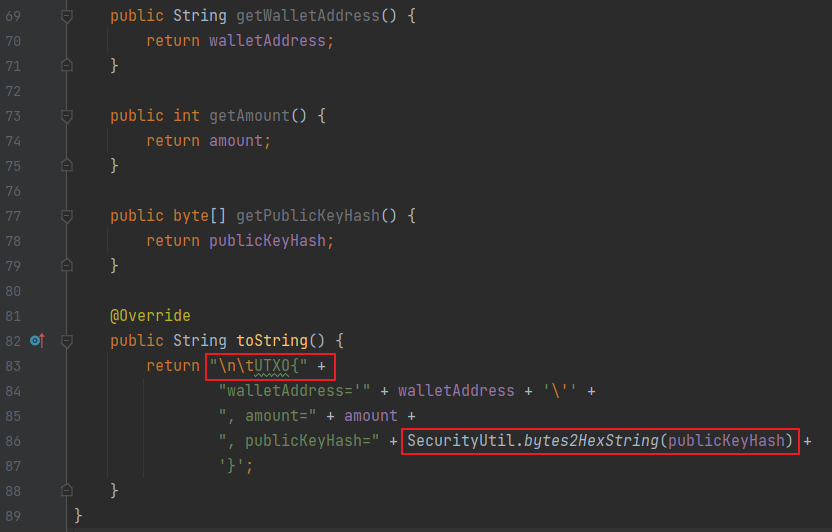


图2-16 UTXO类\_方法\_toString

系统支持UTXO后，钱包就可以计算某个账户的余额，即将账户地址关联的UTXO汇总。回到Account类中，添加一个计算账户金额的方法，如图2-17所示。



图2-17 Account类\_方法\_计算账户余额

该方法的参数是trueUtxos，意指与该账户关联的未被使用的UTXO。而那些已经被使用了的UTXO，我们将其称为fakeUtxos。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**

在实验1中，系统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。回顾实验介绍，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类结构如图2-18所示。



图2-18 Transaction类修改

Alt+Insert快捷键快速生成相应的getter方法和toString方法，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，代码如图2-19所示。

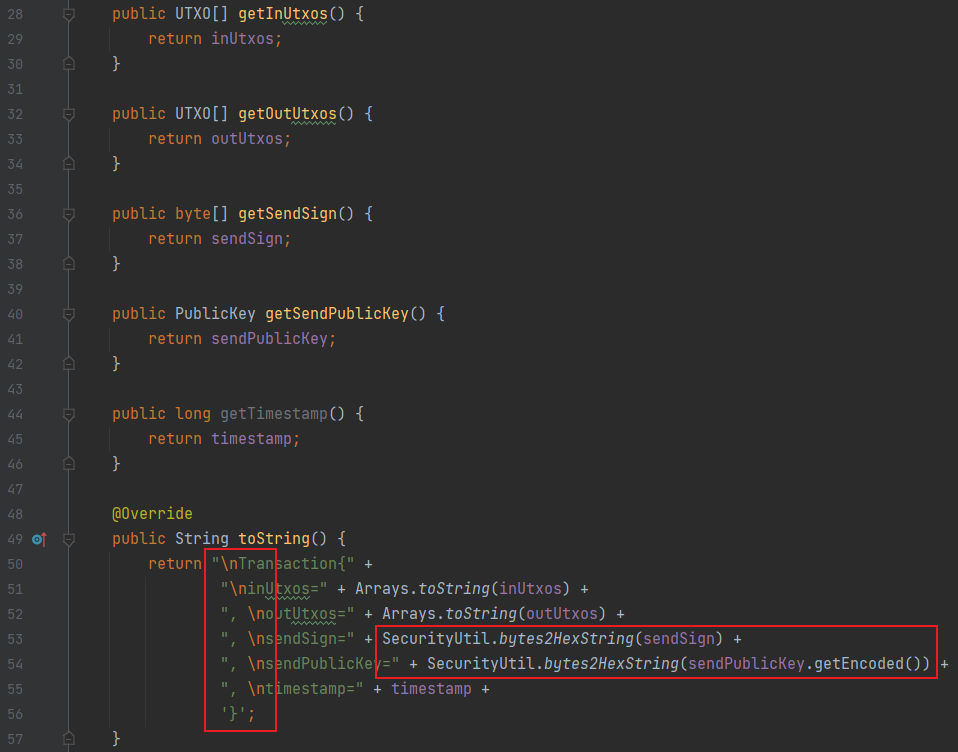


图2-19 Transaction类\_方法\_toString

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。我们在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，如图2-20所示。

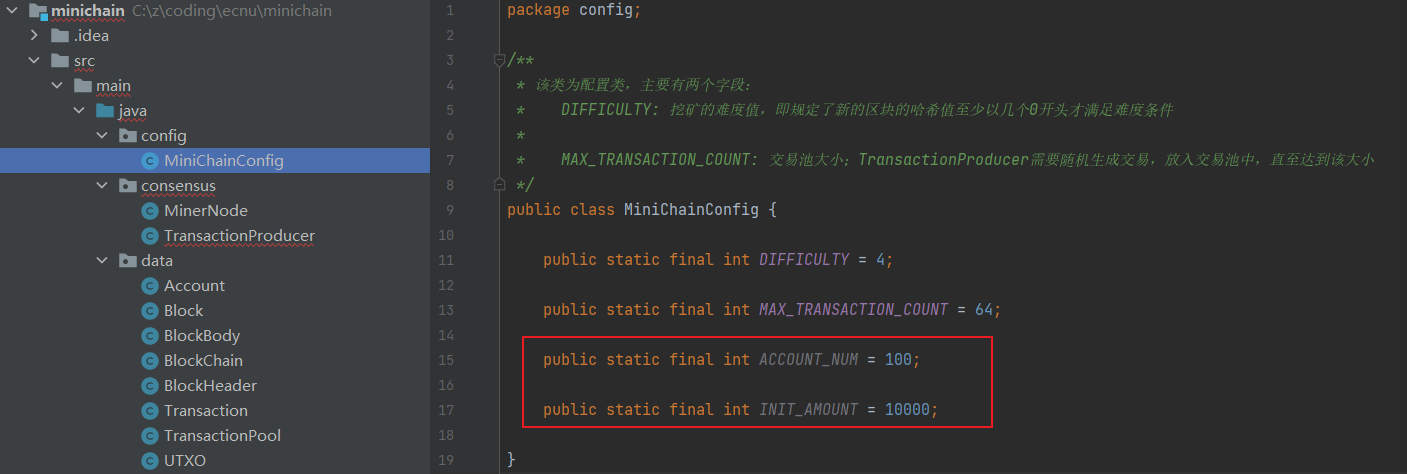


图2-20 MiniChainConfig类修改

随后在BlockChain类中添加Account账户数组，然后修改构造方法，在初始区块中添加为这些账户提供的UTXO，使每个账户在系统启动时就拥有10000比特币，便于后面交易的进行，代码如图2-21所示。



图2-21 BlockChain类构造方法修改

下一步，实现genesisTransactions方法，在BlockChain类中添加相应方法，该方法会创建一批输出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，代码中已经给出了答案，如图2-22所示。

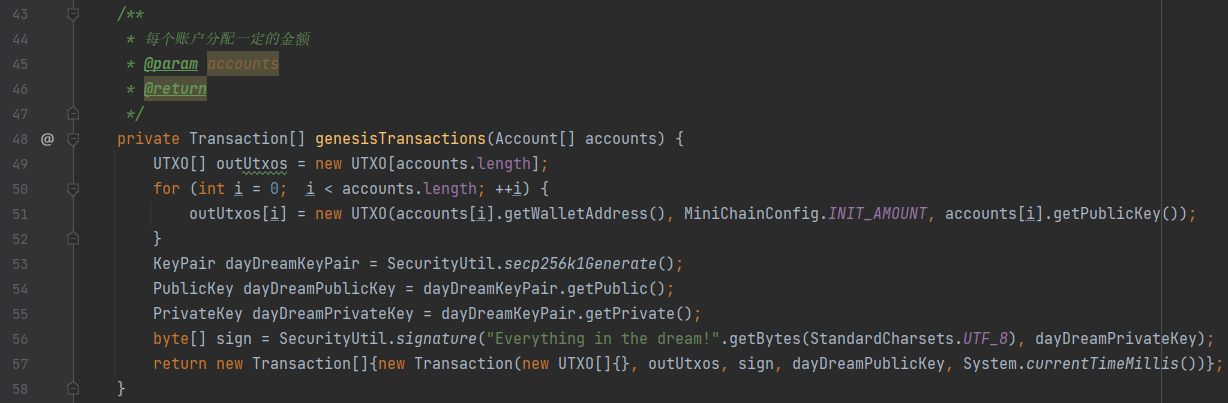


图2-22 BlockChain类\_方法\_genesisTransactions

在BlockChain类中，我们还需要实现一个方法getTrueUtxos，查询出某个账户可使用的UTXO，定义为trueUtxos数组，以便构造后续交易使用。在Account类的getAmount方法中，正需要trueUtxos作为参数。而查询某账户的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关联当前账户的地址。实现代码如图2-23所示。



图2-23 BlockChain类\_方法\_getTrueUtxos

按下Alt+Insert快捷键快速生成accounts的getter方法，如图2-24所示。

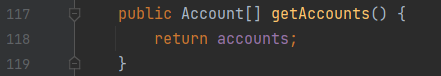


图2-24 BlockChain\_方法\_getAccounts

1. **随机生成一笔交易**

现在，系统有了一定量的账户，并且能查清他们的余额，他们可以在链上开始交易了。实验1中，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交互，故需要添加一个blockchain数据成员并修改其构造方法，如图2-25所示。

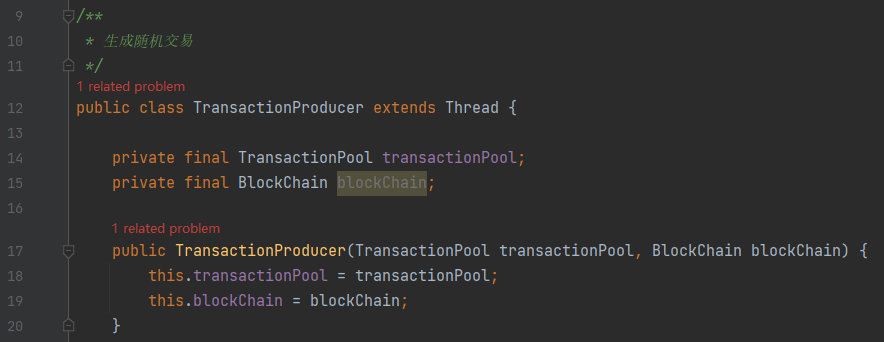


图2-25 TransactionProducer类修改

IDE提示相关错误，点击上图红字跳转到NetWork类，修改相关代码，如图2-26所示。



图2-26 NetWork构造方法修改

在TransactionProduce类中，修改getOneTransaction方法，获取链上的真实账户地址，生成随机的交易额，构造真实的UTXO和交易，并对交易进行签名，代码如图2-27所示。

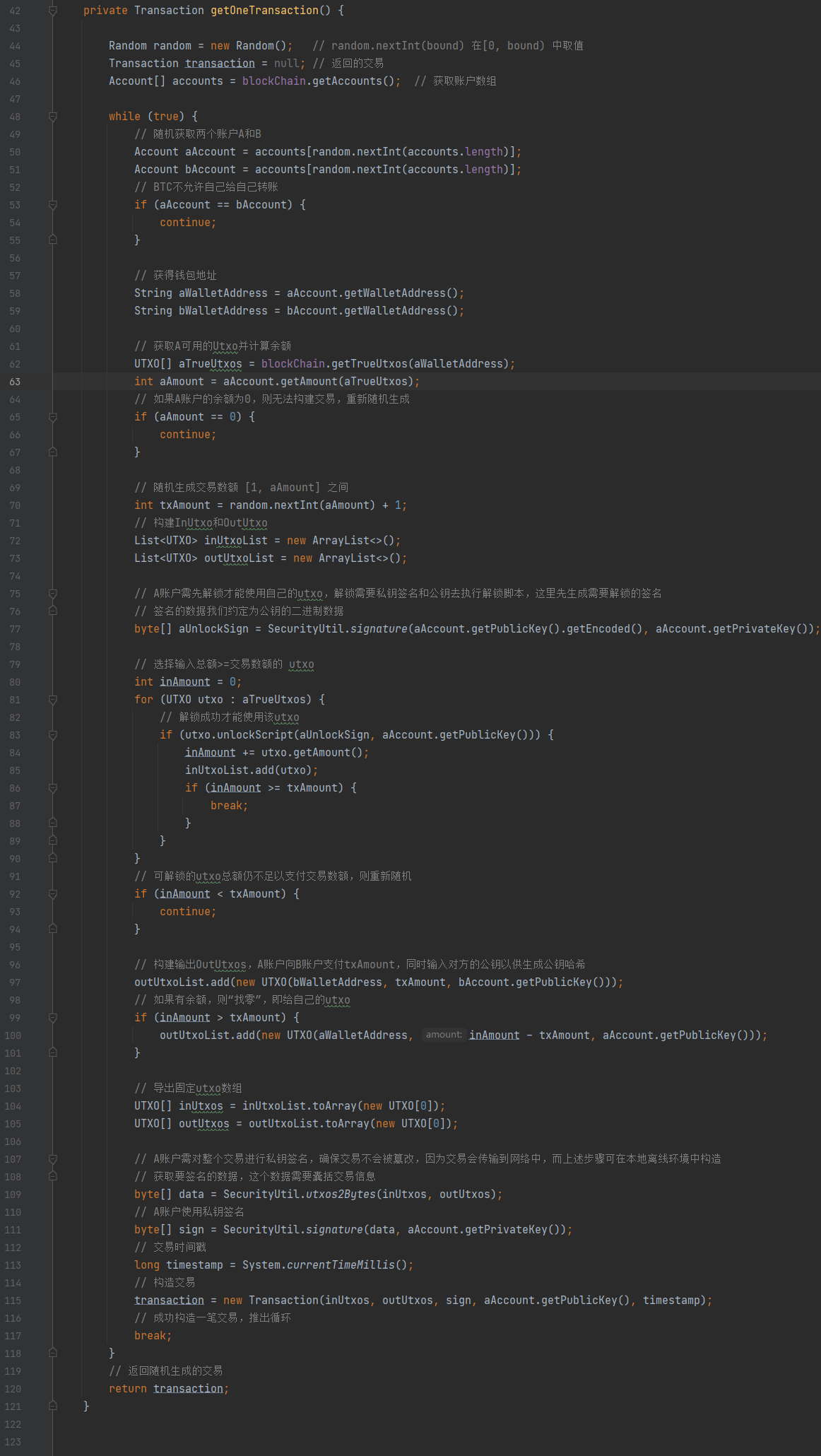


图2-27 TransactionProducer类\_方法\_getOneTransaction

1. **矿工检查工作**

在前面的步骤中，我们实现了交易的签名，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，故在MinerNode类添加如下代码，如图2-28红色方框部分所示。

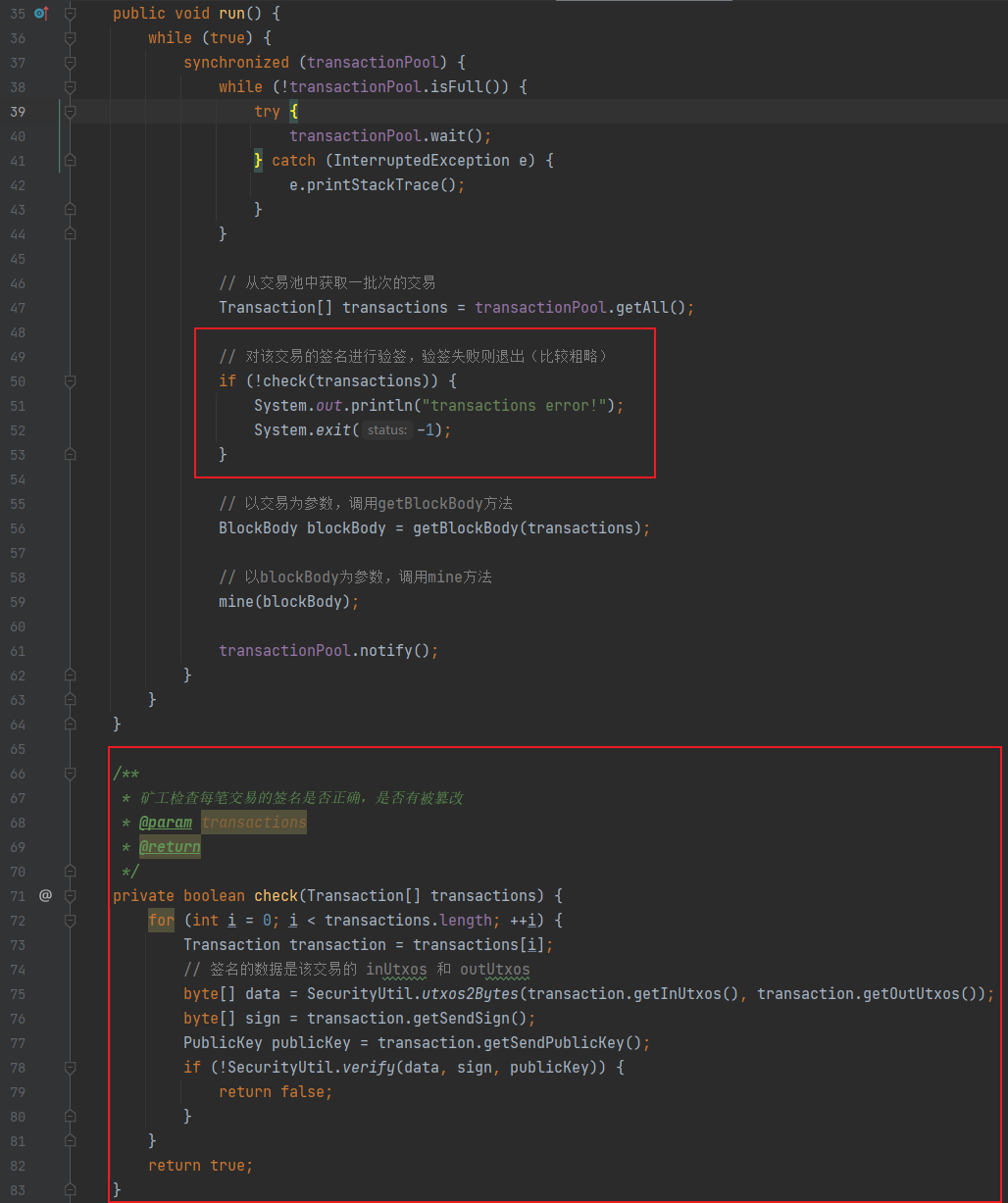


图2-28 MinerNode类修改

1. **运行**

将Test测试代码中因代码修改而出错的地方注释，如图2-29所示。

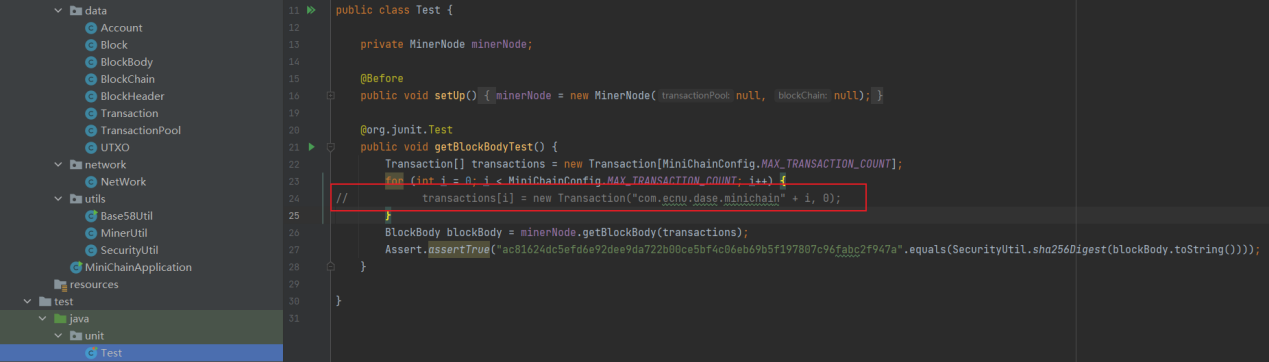


图2-29 注释测试方法

运行MiniChainApplication类中的main方法启动minichain，读者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，如图2-30所示。

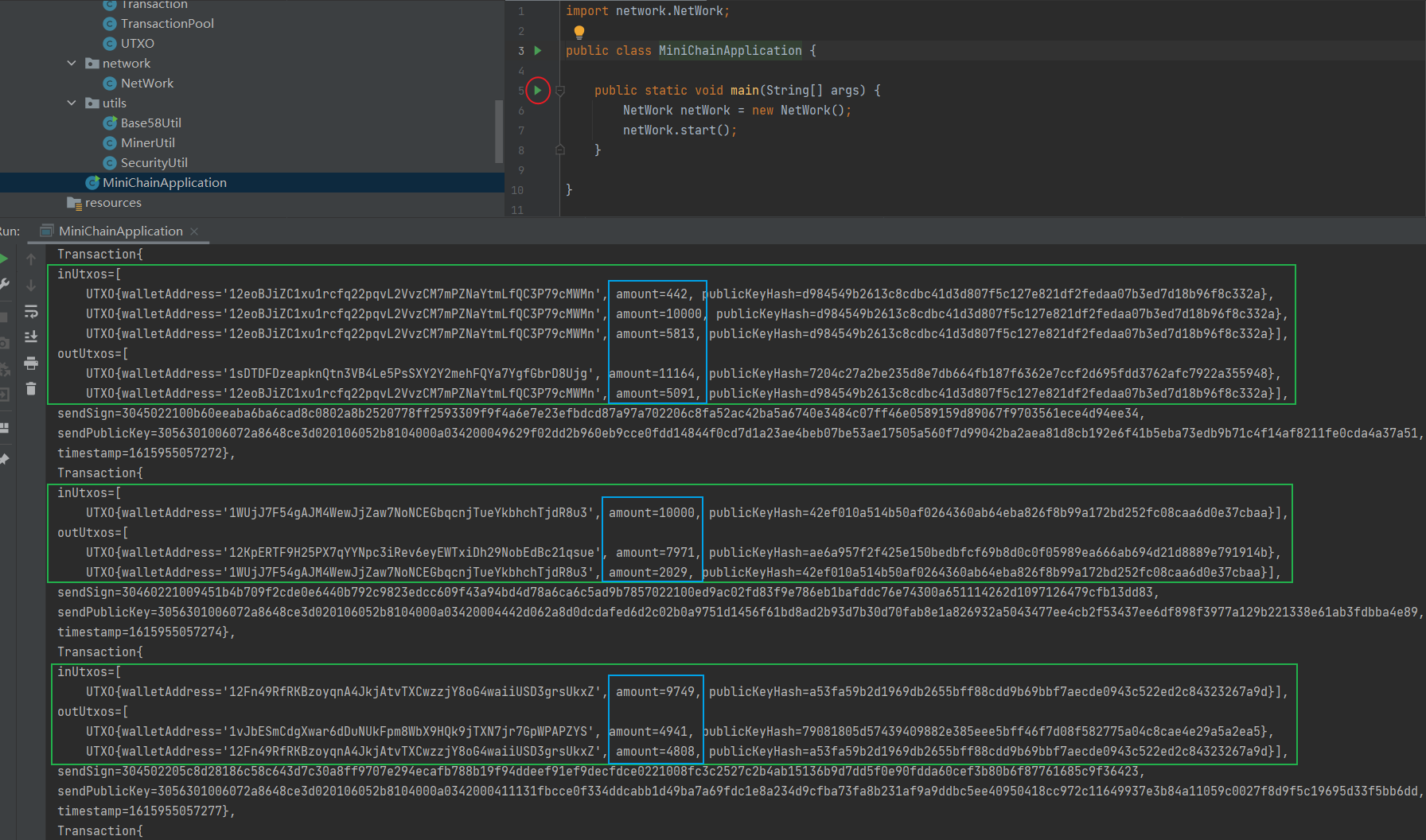


图2-30 运行结果

1. **验证**

为了验证交易中的UTXO是否正确，我们可以在系统每出一个块时统计所有账户的总余额，如果所有账户的总额等于初始账户数×初始账户金额（此处是1000000），就能在一定程度上保证程序运行正确。在BlockChain类中添加统计下所有账户的总余额的方法，如图2-31所示。

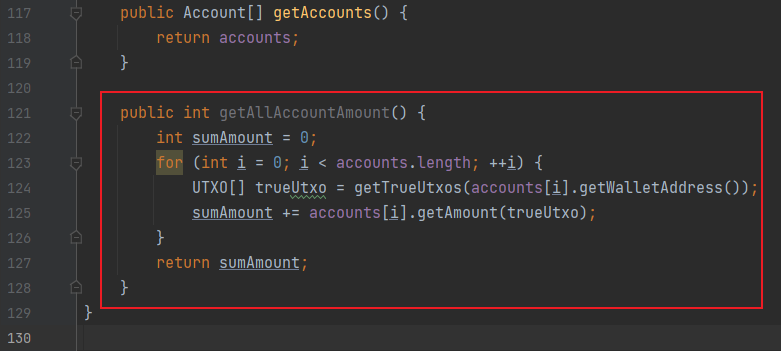


图2-31 BlockChain类\_方法\_getAllAccountAmount

在MineNode类中的run方法添加输出代码，如图2-32所示。

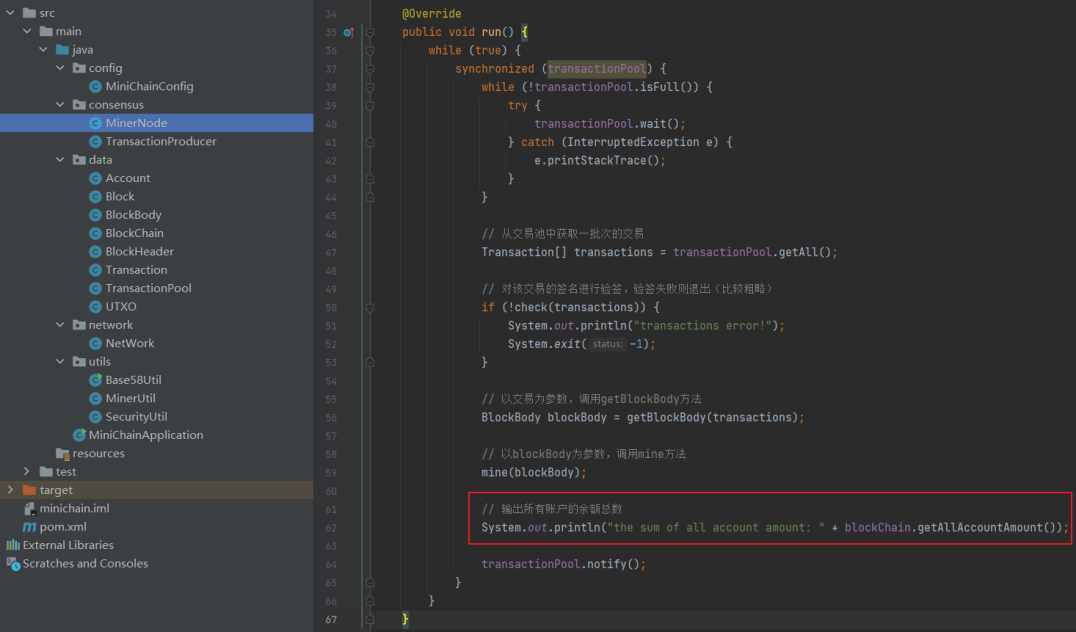


图2-33 输出所有账户的余额总数

重新运行程序，样例输出如图2-33所示。

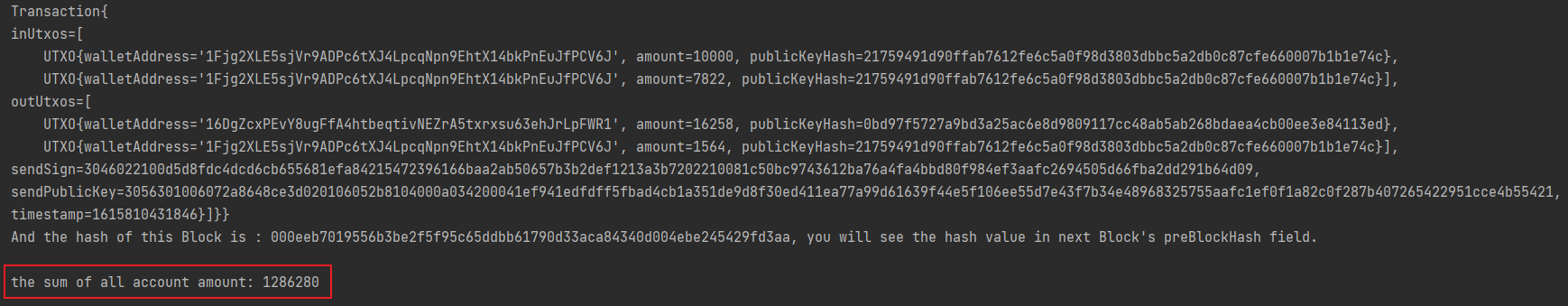


图2-33 样例输出1—所有账户的余额总数

读者会发现总金额与我们预期的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，故会让总金额慢慢变大。在实际的比特币网络中，有更多更严格的检查机制。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），如图2-34所示。

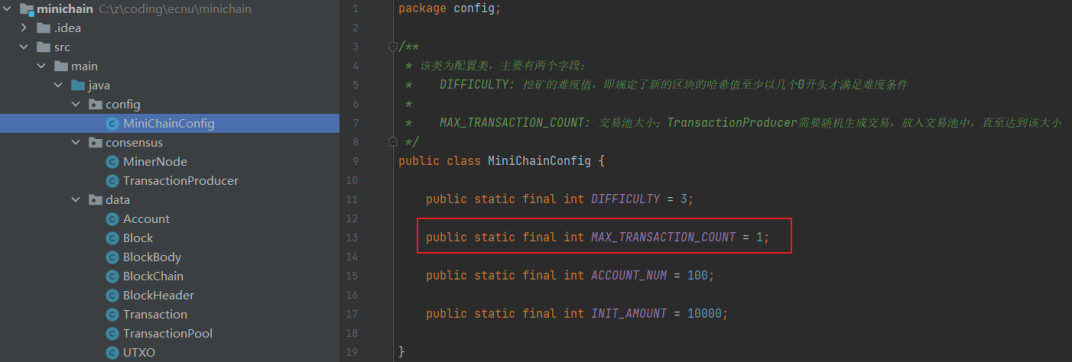
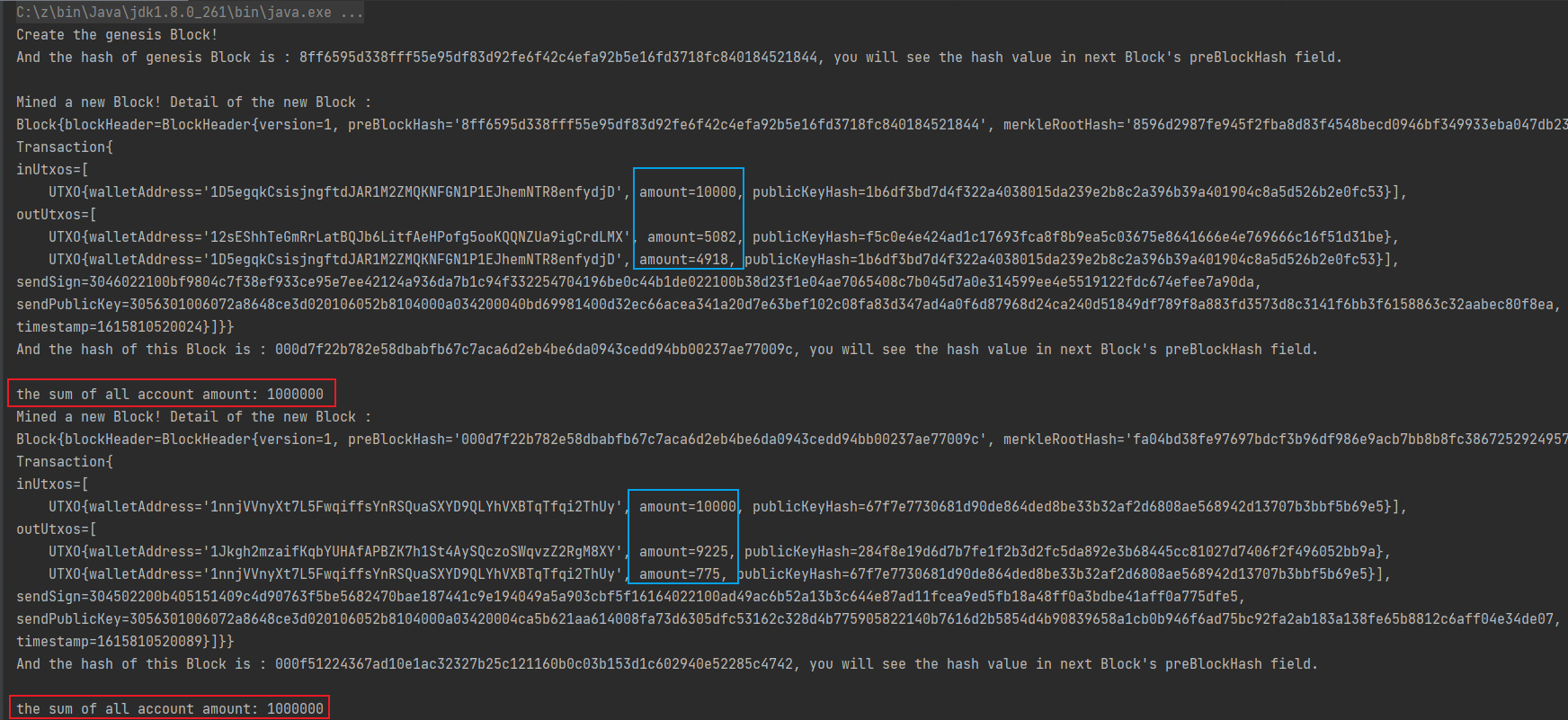


图2-34 修改块的大小

重新运行程序，如图2-35所示，可以看到所有账户的总余额与预期的1000000一致，并且一个交易中的输入UTXO和输出UTXO在金额上时相等的。

图2-35 样例输出2—所有账户的余额总数

**【实验小结】**

本实验从密码学角度切入区块链系统，简要阐述了非对称加密、数字签名和UTXO的基本概念。并以比特币为例，描述了一般区块链系统中私钥、公钥以及地址之间的关系，同时给出了非对称加密在数字签名算法中的工作原理。另外，本实验给出了一个Alice向Bob转账的例子进一步说明比特币如何巧妙地使用UTXO交易模型来解决双花问题。

通过本实验的理论学习与工程实践，读者已经能初步理解常见的密码学技术在区块链系统中的工作原理。但由于篇幅限制，密码学在区块链系统中的其他应用如用于隐私保护的群签名、环签名、盲签名和门限签名，以及用于身份认证的数字证书等，均没有在本实验中进行介绍。感兴趣的读者可以阅读其他相关资料，在本实验提供的简单区块链系统中进行相应代码的实现。

**【习题】**

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。

2. UTXO模型和账户模型的各自优缺点是什么？

3. 在实验完成的基础上，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1000BTC 给账户B(accounts[2])。

**【参考文献】**

1. 陈少真. 密码学基础[M]. 科学出版社, 2008.
2. 王厚涛.SSL VPN安全技术研究及改进[D].北京邮电大学,2011.
3. 何昊.区块链架构之美[M].电子工业出版社,2021.
4. 汪朝晖, 张振峰. SM2 椭圆曲线公钥密码算法综述[J]. 信息安全研究, 2016.
5. 秦波, 陈李昌豪, 伍前红, 等. 比特币与法定数字货币[J]. 密码学报, 2017, 4(2): 176-186.