Как я написал свой blockchain

Для начала загуглил и нашёл два руководства, но оба написаны для Python (который я почти не знаю), да ещё и с дополнительными библиотеками. А так как сейчас активно изучаю программирование под Android и Java – решил создать проект на Java, сразу разместив его на BitBucket (который в этот день хотел меня о чём-то предупредить разместив радужный логотип)

# Создание проекта

Итак, меньше слов и больше дела. Поехали. В основном модуле создаём объект класса Block:

**public class** Main {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"Hello, blockchain!"**);  
 Block block = **new** Block();  
 }  
}

А в самом классе Block будем создавать блокчейн. Пока только заглушки, по которым видно, что собираемся дальше сделать. В самом объекте 2 поля: chain – наша цепочка блоков, которую будем строить и transaction – цепочка транзакций, которые будут храниться.

**import** java.util.LinkedList;  
  
**public class** Block {  
  
 **private** LinkedList<String> **chain**;  
 **private** LinkedList<String> **transaction**;  
  
 **public** Block(){  
 **this**.**chain** = **new** LinkedList<String>();  
 **this**.**transaction** = **new** LinkedList<String>();  
 }  
  
 **public** LinkedList<String> newBlock(){  
 *//* ***TODO: Создаём новый блок и добавляем его к цепочке* return new** LinkedList<String>();  
 }  
  
 **public** LinkedList<String> newTransaction(){  
 *//* ***TODO: Создаём новую транзакцию и добавляем её в список транзакций* return new** LinkedList<String>();  
 }  
  
 **public static** String hash(Block block){  
 *//* ***TODO: Формируем хеш блока и возвращаем его* return ""**;  
 }  
  
 **public** Block last(){  
 *//* ***TODO: Возвращаем последний блок в цепочке* return new** Block();  
 }  
}

А так же добавил соответствующие unit-тесты заглушки. Хочу заодно попробовать метод TTD.

**import** org.junit.After  
**import** org.junit.Assert  
**import** org.junit.Before  
**import** org.junit.Test  
  
**class** MainTest **extends** GroovyTestCase {  
 **private** Main **main**;  
  
 @Before  
 **void** initTest() {  
 **main** = **new** Main();  
 }  
  
 @After  
 **void** afterTest() {  
 **main** = **null**;  
 }  
  
 @Test  
 **void** testMain() {  
 Assert.*assertTrue*(**true**);  
 }  
  
 @Test(timeout = 500l)  
 **void** timeStampTest() {  
 **while** (**true**);  
 }  
}

И соответствующие тесты для Block

**import** org.junit.After  
**import** org.junit.Assert  
**import** org.junit.Before  
**import** org.junit.Test  
  
**class** BlockTest **extends** GroovyTestCase {  
 **private** Block **block**;  
  
 @Before  
 **void** initTest() {  
 **block** = **new** Block();  
 }  
  
 @After  
 **void** afterTest() {  
 **block** = **null**;  
 }  
  
 @Test  
 **void** testBlock() {  
 Assert.*assertTrue*(**true**);  
 }  
  
 @Test(timeout = 500l)  
 **void** timeStampTest() {  
 **while** (**true**);  
 }  
  
 @Test  
 **void** testNewBlock() {  
 Assert.*assertTrue*(**true**);  
 }  
  
 @Test  
 **void** testNewTransaction() {  
 Assert.*assertTrue*(**true**);  
 }  
  
 @Test  
 **void** testHash() {  
 Assert.*assertTrue*(**true**);  
 }  
  
 @Test  
 **void** testLast() {  
 Assert.*assertTrue*(**true**);  
 }  
}

Все тесты прошли успешно – можно продолжать.

# Заполнение класса блокчейна

Когда начал дальше разбираться – пришлось основной класс переименовать в Blockchain, добавить 2 новых класса Block и Message. А так же решил переименовать везде Transaction на Message – чтобы не загромождать код заумными словами. И вот, что получилось:

**Blockchain.java**

**import** java.nio.charset.StandardCharsets;  
**import** java.security.MessageDigest;  
**import** java.security.NoSuchAlgorithmException;  
**import** java.util.Base64;  
**import** java.util.LinkedList;  
  
**public class** Blockchain {  
  
 **private** LinkedList<Block> **chain**;  
 **private** LinkedList<Message> **messages**;  
  
 **public** Blockchain(){  
 **this**.**chain** = **new** LinkedList<Block>();  
 **this**.**messages** = **new** LinkedList<Message>();  
 **this**.newBlock(100, **""**);  
 }  
  
 **public** Block newBlock(**int** proof, String prevHash){  
 Block block = **new** Block(**this**.**chain**.size(), **this**.**messages**, proof, prevHash);  
 **this**.**messages** = **new** LinkedList<Message>();  
 **this**.**chain**.add(block);  
 **return** block;  
 }  
  
 **public int** newMessage(String from, String to, **float** amount){  
 Message message = **new** Message(from, to, amount);  
 **this**.**messages**.add(message);  
 **return this**.**messages**.size();  
 }  
  
 **public static** String hash(Block block) **throws** NoSuchAlgorithmException {  
 String text = block.toString();  
 MessageDigest digest = MessageDigest.*getInstance*(**"SHA-256"**);  
 **byte**[] hash = digest.digest(text.getBytes(StandardCharsets.***UTF\_8***));  
 **return** Base64.*getEncoder*().encodeToString(hash);  
 }  
  
 **public** Block last(){  
 **return this**.**chain**.getLast();  
 }  
}

**Block.java**

**import** java.util.Date;  
**import** java.util.LinkedList;  
  
**public class** Block **extends** Object{  
 **public** Date **timestamp**;  
 **public int index**;  
 **public** LinkedList<Message> **messages**;  
 **public int proof**;  
 **public** String **prevHash**;  
  
 **public** Block(**int** index, LinkedList<Message> messages, **int** proof, String prevHash){  
 **this**.**timestamp** = **new** Date();  
 **this**.**index** = index;  
 **this**.**messages** = messages;  
 **this**.**proof** = proof;  
 **this**.**prevHash** = prevHash;  
 }  
}

**Message.java**

**public class** Message {  
 **public** String **from**; *// Адрес отправителя* **public** String **to**; *// Адрес получателя* **public float amount**; *// Количество* **public** Message(String from, String to, **float** amount){  
 **this**.**from** = from;  
 **this**.**to** = to;  
 **this**.**amount** = amount;  
 }  
}

Ссылки

1. Основная статья по основам создания blockchain: <https://proglib.io/p/learn-blockchains-by-building-one/>
2. Дополнительная статья по созданию blockchain: <https://habr.com/company/otus/blog/342974/> (перевод статьи <https://bigishdata.com/2017/10/17/write-your-own-blockchain-part-1-creating-storing-syncing-displaying-mining-and-proving-work/> )
3. Подключение unit-тестов в IDEA: <http://qaru.site/questions/156160/setting-up-junit-with-intellij-idea>
4. Написание unit-тестов в IDEA: <http://javastudy.ru/junit/junit-hello-world/>
5. Получение SHA256: <https://stackoverflow.com/questions/5531455/how-to-hash-some-string-with-sha256-in-java>