# Как я написал свой blockchain

## Введение

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

## Main.java

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();  
 }  
}

## Block.java

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();  
 }  
}

## MainTest.groovy

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**);  
 }  
}

## BlockTest.groovy

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 также решил переименовать везде Transaction на Message – чтобы не загромождать код заумными словами. То, как работает код и какие unit-тесты написал - расскажу позже, когда буду дорабатывать unit-тесты. 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;  
 }  
}

# Добавление документации

## Правила написания документации

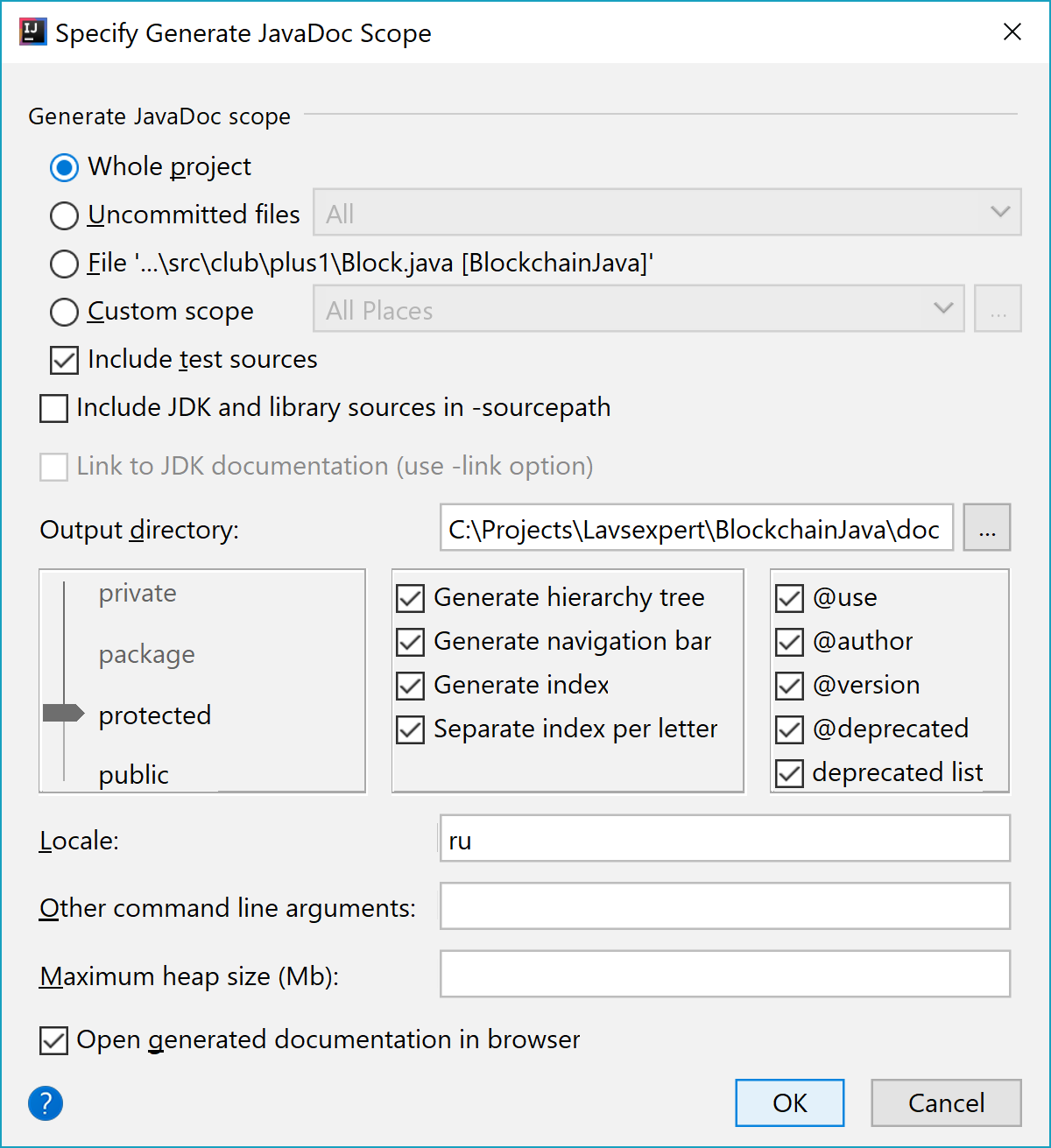
Далее изучал то, как правильно пишется JavaDoc. Эффективная Javadoc должна содержать описания того:

* Как классы взаимосвязаны друг с другом
* Как методы влияют на состояние объекта
* Как методы сообщают вызывающим устройствам о ситуациях ошибки, и о каких ошибках они могут оповещать
* Как класс справляется с использованием в многопоточном приложении
* Домен аргументов метода и множество их возвращаемых значений

Плохая документация - или что еще хуже, ее отсутствие - может привести хороший код в состояние непригодности или одноразового пользования. Потратив немного дополнительного времени на документацию, вы сэкономите несчетное количество часов ваших пользователей (а может и своих).

## Выгрузка документации

В итоге добавил JavaDoc во все классы. Cделал выгрузку JavaDoc (в IDEA: Tools -> Generate JavaDoc) Теперь есть документация в виде html.



## Main.java

*/\*\*  
 \* Основной класс программы, через который идёт взаимодействие с другими классами и консолью  
 \** ***@author*** *lavsexpert  
 \** ***@version*** *1.0  
 \*/***public class** Main {  
  
 */\*\*  
 \* Главная функция программы. Создаёт объект <b>блокчейн(Blockchain)</b>  
 \** ***@param args*** *параметры командной строки  
 \*/* **public static void** main(String[] args) {  
 System.***out***.println(**"Hello, blockchain!"**);  
 Blockchain block = **new** Blockchain();  
 }  
}

## Blockchain.java

**import** java.nio.charset.StandardCharsets;  
**import** java.security.MessageDigest;  
**import** java.security.NoSuchAlgorithmException;  
**import** java.util.Base64;  
**import** java.util.LinkedList;  
  
*/\*\*  
 \* Класс описывает объекты <b>Blockchain(Цепочка блоков)</b> - которые являются основными объектами проекта.  
 \* Так же в класе <b>Blockchain</b> используются классы:  
 \* <b>Block</b> в качестве элемента <b>LinkedList</b> в поле <b>chain</b>;  
 \* <b>Message</b> в качестве элемента <b>LinkedList</b> в поле <b>messages</b>.  
 \* Ошибки в классе пока не обрабатываются и не перехватыаются(кроме <b>NoSuchAlgorithmException</b> в методе <b>hash</b>).  
 \** ***@author*** *lavsexpert  
 \** ***@version*** *1.0  
 \*/***public class** Blockchain {  
  
 **private** LinkedList<Block> **chain**; *// Список блоков в цепочке* **private** LinkedList<Message> **messages**; *// Список сообщений(транзакций)  
  
 /\*\*  
 \* Конструктор объекта <b>Blockchain (Цепочка блоков)</b>  
 \* Создаёт цепочку с пустыми списками сообщений и блоков. Добавляет корневой блок с доказательством.  
 \*/* **public** Blockchain(){  
 **this**.**chain** = **new** LinkedList<Block>();  
 **this**.**messages** = **new** LinkedList<Message>();  
 **this**.newBlock(100, **""**);  
 }  
  
 */\*\*  
 \* Создание нового блока в блокчейне с пустым списком сообщений  
 \** ***@param proof*** *доказательство блока (пока не проверяется)  
 \** ***@param prevHash*** *хеш предыдущего блока  
 \** ***@return*** *новый объекта класса Block(Блок)  
 \*/* **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;  
 }  
  
 */\*\*  
 \* Создание нового сообщения в текущем блоке  
 \** ***@param from*** *отправитель сообщения  
 \** ***@param to*** *получатель сообщения  
 \** ***@param amount*** *количество  
 \** ***@param comment*** *комментарий  
 \** ***@return*** *номер сообщения в блоке  
 \*/* **public int** newMessage(String from, String to, **float** amount, String comment){  
 Message message = **new** Message(from, to, amount, comment);  
 **this**.**messages**.add(message);  
 **return this**.**messages**.size();  
 }  
  
 */\*\*  
 \* Поучение хеша блока  
 \** ***@param block*** *блок, для которого надо получить кэш  
 \** ***@return*** *строка с хешем блока  
 \** ***@throws*** *NoSuchAlgorithmException может появиться ошибка при создании хеша  
 \*/* **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);  
 }  
  
 */\*\*  
 \* Получение последнего блока в блокчейне  
 \** ***@return*** *последний блок  
 \*/* **public** Block last(){  
 **return this**.**chain**.getLast();  
 }  
}

## Block.java

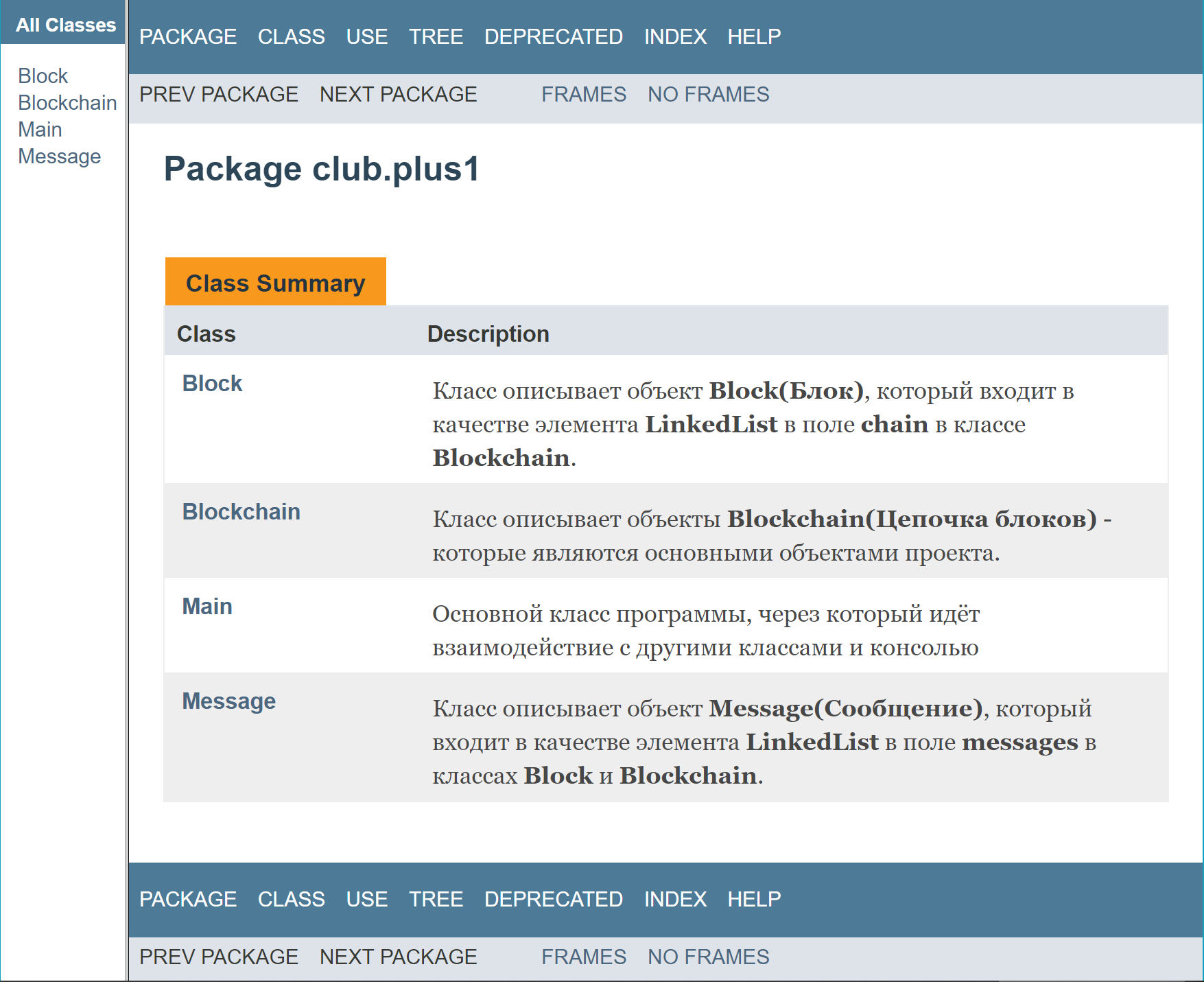
**import** java.util.Date;  
**import** java.util.LinkedList;  
  
*/\*\*  
 \* Класс описывает объект <b>Block(Блок)</b>, который входит в качестве элемента <b>LinkedList</b>  
 \* в поле <b>chain</b> в классе <b>Blockchain</b>. Так же в класе <b>Block</b> используется класс <b>Message</b>  
 \* в качестве элемента LinkedList в поле <b>messages</b>.  
 \* Пока в классе все поля публичны, что не безопасно.  
 \* Ошибки в классе пока не обрабатываются и не перехватыаются.  
 \** ***@author*** *lavsexpert  
 \** ***@version*** *1.0  
 \*/***public class** Block {  
 **public** Date **timestamp**; *// Отметка времени* **public int index**; *// Код(индекс) блока* **public** LinkedList<Message> **messages**;*// Список сообщений(транзакций)* **public int proof**; *// Доказательство* **public** String **prevHash**; *// Хеш предыдущего блока  
  
 /\*\*  
 \* Конструктор объекта <b>Block(Блок)</b> в блокчейне  
 \** ***@param index*** *код(индекс) блока  
 \** ***@param messages*** *список сообщений(транзакций)  
 \** ***@param proof*** *доказательство  
 \** ***@param prevHash*** *хеш предыдущего блока  
 \*/* **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

*/\*\*  
 \* Класс описывает объект <b>Message(Сообщение)</b>, который входит в качестве элемента <b>LinkedList</b>  
 \* в поле <b>messages</b> в классах <b>Block</b> и <b>Blockchain</b>.  
 \* Пока в классе все поля публичны, что не безопасно.  
 \* Ошибки в классе пока не обрабатываются и не перехватыаются.  
 \** ***@author*** *lavsexpert  
 \** ***@version*** *1.0  
 \*/***public class** Message {  
 **public** String **from**; *// Адрес отправителя* **public** String **to**; *// Адрес получателя* **public float amount**; *// Количество* **public** String **comment**; *// Комментарий  
  
 /\*\*  
 \* Конструктор объекта <b>Message(Сообщение|транзакция)</b> в блокчейне  
 \** ***@param from*** *адрес отправителя  
 \** ***@param to*** *адрес получателя  
 \** ***@param amount*** *количество  
 \** ***@param comment*** *комментарий  
 \*/* **public** Message(String from, String to, **float** amount, String comment){  
 **this**.**from** = from;  
 **this**.**to** = to;  
 **this**.**amount** = amount;  
 **this**.**comment** = comment;  
 }  
}

## Документация в html

В итоге документация выглядела так:



# Приложения

## 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>
6. Как писать JavaDoc: <https://www.ibm.com/developerworks/ru/library/j-jtp0821/index.html>
7. Пример кода с JavaDoc: <http://www.fandroid.info/dokumentirovanie-koda/>