ماهوالـ BLOCKCHAIN ؟

إعداد

د. سلطان بن سعود القحطاني

أستاذ مساعد بكلية علوم الحاسب والمعلومات

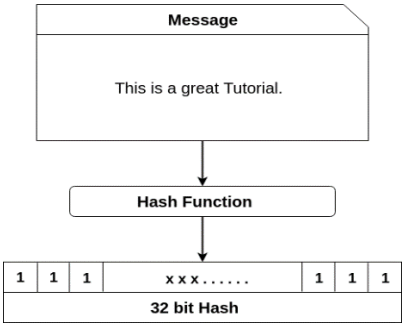
جامعة الإمام محمد بن سعود الإسلاميه

1. مقدمة

أود ان اوضح بأن تقنية الـ Blockchain هي مفهوم بسيط وسهل جدا لمن لديهم مفهوم هندسة البرمجيات وتطوير البرامج. الـ Blockchain هي مفهوم من مفاهيم تراكيب البيانات (Data Structure) وبالطبع مفهوم الـ Cryptocurrency يعتبر مفهوم معقد ولاكن تقنية الـ Blockchain ليست السبب في ذلك. السبب في ان مفهوم الـ Cryptocurrency معقده هو ما تحتويه من خوارزميات (Algorithms) خاصة بالتنقيب (Mining) ونقصد بالتنقيب هنا هو الـ Bitcoin mining,ايضا في تعقيد مفهوم الـ Cryptocurrency هو ما يرتبط بها من العمليات الحسابيه وتوزيعها, مفهوم الـ Ledger (او مايعرف بالمحفظة الماليه بالمفهوم العام).

قبل ان نبدأ في شرح تقنية الـ Blockchain وكيفية عملها, تحتاج ان تتعرف على مفهوم الـ Hash ولماذا هو مهم في تقنية الـ Blockchain.

الـ Hash في الحقيقه هو مفهوم مشابه لمفهوم التوقيع الرقمي (Digital Signature) أو كمن تعطي شيء صفة خاصه به تميزه عن غيره مثل رقم بطاقة الأحوال (Digital ID). مثلا ممكن نأخذ ملف فلم أو نص أو صوره ونعمل له Hash ويصبح لهذا الملف رقم خاص به أو Digital ID, ايضا ممكن نأخذ بريد إلكتروني ونعمل له Hash ويصبح هذا الهاش هو الرقم الخاص بهذا البريد الإلكتروني. مثال الصورة التالية:



عبارة عن ملف نصي, عملنا له Hash وسيخرج رقم عشوائي وهذا الرقم هو Digital ID لهذا الملف النصي واللذي يكون مميز عن غيره. لكي اتحقق بأن هذا الملف النصي صحيح لابد ان يكون الـ Digital ID مطابق. لو اخذنا المثال التالي:

```
String[] list1 = {"a", "b", "c"};
String[] list2 = {"a", "b", "c"};
```

عباره قائمتن (two lists), القائمه الأولى (list1) تحتوي على "a", "b", "c" بينما القائمه الثانيه (list2) تحتوي على "a", "b", "c". محتوى القائمتين عباره عن بيانات وكل قائمه عباره عن بيانات خاصه بها لكن المحتوى متشابه. لو قمنا بصناعة Digital ID خاص بها ماذا سيحدث؟

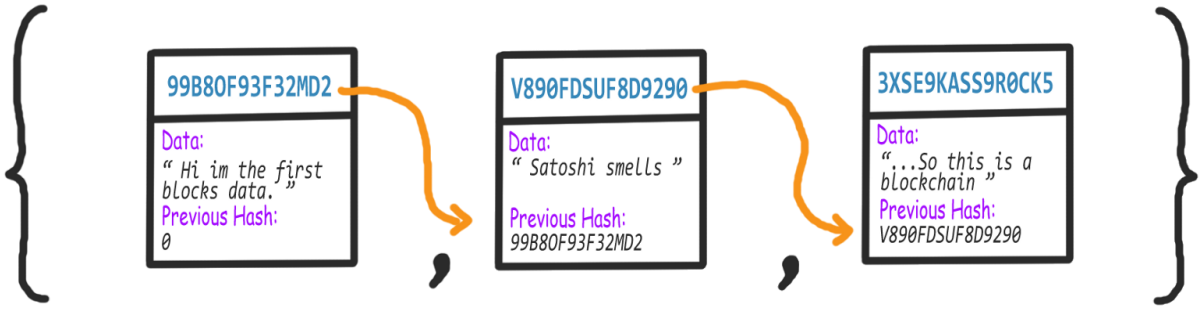
```
String[] list1 = {"a", "b", "c"};
String[] list2 = {"a", "b", "c"};
System.out.println(Arrays.hashCode(list1));
System.out.println(Arrays.hashCode(list2));
```

نلاحظ ان المخرجات لكل عملية طباعه ستكون متطابقه على النحو التالي: **126145** وهو يمثل الـ Digital ID أو الـ Digital signature للبيانات في الـ list1 و **126145** لـ list2 نفس الشيء. لو قمنا بتعديل محتوى list2 مثلا إلى:

```
String[] list1 = {"a", "b", "c"};
String[] list2 = {"a", "b", "c"};
System.out.println(Arrays.hashCode(list1));
System.out.println(Arrays.hashCode(list2));
```

ستكون المخرجات مختلفه على النحو التالي: 126145 لـ list1 و 126642 لـ list2, فبالتالي إذا عملنا تغيير بسيط على أحد المحتويات في القئمتين الموضحه في المثال سوف نحصل على Digital signature مختلف تماما عن السابق, وهذا هو الأساس المبني عليه تقنية الـ Blockchain.

لأن الـ Blockchain هي عبارة عن قائمه (list) من الـ Blocks أو في الأساس سلسله (chain) من الـ Blocks وكل Block يحتوي على التالي: 1- مجموعه من الـ Transactions (العمليات اللتي تمت في هذا الـ Block) و 2- Digital ID أو Signature خاص فيه و 3- Digital signature خاص بالـ Block اللذي قبل.كما هو مبين في الصوره التاليه:

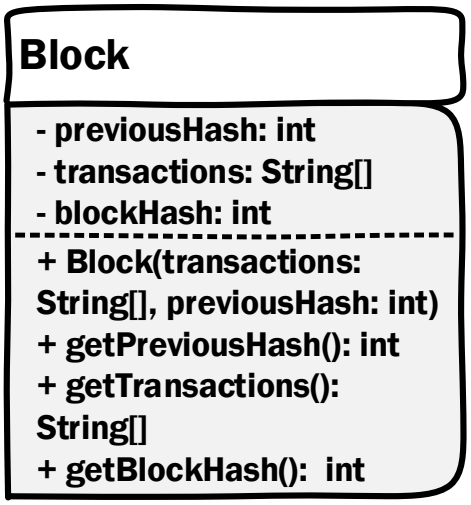


أي بمعنى أن الـ Blocks في السلسله معتمدين على بعض, فبالتالي, الـ Digital ID لأي Block في السلسلة معتمد على الـ Digital ID للبلوك اللذي قبله. إذا غيرنا في اي محتوى Block (اي غيرنا على البيانات الخاصه بالـ Block) في السلسله, فهذا يعني اننا اثرنا على جميع الـ Blocks المرتبطه في السلسله.

بعد ماتعرفنا على مفهوم الـ Hash وارتباطه بتقنية الـBlockchain, في القسم التالي سنبني برنامج بسيط يشرح كيفية برمجة الـ Blockchain بإستخدام لغة Java.

2. تطبيق الـ Blockchain بأستخدام لغة Java ؟

في البداية اول شيء نتحدث عنه في تركيبة الـ Blockchain هو الـ Block وماذا يحتوي. في هذا المثال كما هو موضح في الصوره التاليه:



في هذا المثال, الـ Block اللذي سنبنيه بسيط جدا ويحتوي على ثلاث اشياء رئيسية: 1- قائمة من العمليات (transactions) اللتي تمت من خلال هذا الـ Block, 2- قيمة الـ Hash السابق (قيمة الـ Hash الخاص بالـ Block المرتبط بالـ Block الحالي), 3- الـ Hash الخاص بالـBlock الحالي. بالنسبه للعنصر رقم 3, القيمه الخاصه بهذا الـ Hash ستكون مبنية على قيمة الـ list و الـ Hash السابق, اي ان قيمة الـ Digital ID ستكون محسوبه بناء على العمليات اللتي تمت في الـ Block الحالي و اضافة على قيمة الـ Hash الخاص بالـ Block السابق. فإذا قام شخص ما بتغيير محتوى Block السابق فهذا يعني أن الـ Block الحالي سوف يتغير بناء على التغييرات السابقه, وايضا الـ Block اللذي يليه سوف يتغير وهكذا. لنتعرف سوية على كيفية عمل هذا الكلام النظري بـ code يترجم هذا المفهوم.

```
public class Block {

    private int previousHash;
    private String[] transactions;
    private int blockHash;
    ...
}
```

نصنع Class ونطلق عليه اسم Block ويحتوي هذا الـ Class على العناصر التاليه: previousHash, transactions, blockHash. الـ transactions في هذا المثال فقط استخدمنا مصفوفه من النصووص (Strings) لتوضيح المفهوم ومحاولة صنع Blockchain بمثال بسيط. بينما في الأمثله الحقيقه والتطبيقات الخاصه بالـ Blockchain فالـ transactions تكون اكبر من مجرد مصفوفة نصوص, بل تتكون من تراكيب بيانات واشياء تعتمد على طبيعة البرنامج او النظام اللذي صمم له الـ Blockchain.

عند بناء Objects لـ Block معين, نطبق المفوم المرتبط بالعنصر رقم 3, وستكون دالة الـconstructor كالتالي:

```
public Block(int previousHash, String[] transactions) {
        this.previousHash = previousHash;
        this.transactions = transactions;

        Object[] contents = {Arrays.hashCode(transactions), previousHash};
        this.blockHash = Arrays.hashCode(contens);
}
```

نلاحظ هنا انه عند انشاء Block معين, يكون الـ Hash الخاص به عبارعن المحتوى (contents) الخاص بالـ Block نفسه واللذي هو عباره عن الـ Hash الخاص بالـ transactions (Arrays.*hashCode*(transactions)) اللذي تم في هذا الـ Block و قيمة الـ Hash الخص بالـBlock السابق (previousHash). وبالتالي أي تغيير في احدى القيميتين سيحدث تأثير على قيمة الـ Blocks المرتبطه بهذا الـ Block. اخيرا, لكل Block يوجد مجموعة دوال لإسترجاع المعلومات الخاصه بالـ Block وهي: getBlockHash(),getTransaction(),getPreviousHash() وجميعها دوال فقط تستخدم في حالة البناء للـ Blockchain. وبالتالي يصبح شكل الـ Class Block بعد بناء جميع الدوال والعناصر المكونه للـ Block كالتالي:

```
public class Block {

    private int previousHash;
    private String[] transactions;

    private int blockHash;

    public Block(int previousHash, String[] transactions) {
        this.previousHash = previousHash;
        this.transactions = transactions;

        Object[] contents = {Arrays.hashCode(transactions), previousHash};
        this.blockHash = Arrays.hashCode(contens);

    }

    public int getPreviousHash() {
        return previousHash;
    }

    public String[] getTransaction() {
        return transactions;
    }

    public int getBlockHash() {
        return blockHash;
    }
}
```

بعد أن انتهينا من بناء class Block, تأتي الان طريقة بناء الـ Blockchain, واللتي هي عباره عن سلسله (chain) من الـBlocks. اي اننا نحتاج نوع من انواع تراكيب البيانات (Data structure) تساعدنا في بناء هذه السلسله. في لغة جافا سنستخدم ArrayList لسبب بسيط هو سهولة الإستخدام والتعامل معها وبإمكانك تستخدم LinkedList, Vecotr وغيرها, اهم شيء لابد ان تكون بيانات على شكل سلسله. ولتعريف الـ ArrayList نقوم بالتالي:

```
ArrayList<Block> blockchain = new ArrayList();
```

كما نلاحظ بأن الـblockchain هو عباره عن object من نوع مصفوفه متسلسله وكل عنصر فيها يمثل Block لذلك عملنا ArrayList<Block>.

ذكرت سابقا بأن كل Block مرتبط في Block اخر مما يكون شكل سلسلة, وايضا بناء الـ Hash لكل Block مرتبط بالـBlcok السابق, لكن السؤال هنا هو ماذا عن الـ Block الأول في السلسلة؟ كيف يتم بناء الـ Hash الخاص به إن لم يكون هناك قيمه لـ Hash سابقه. بالنسبة للـBlock الأول في الـ Blockchain يطلق عليه Genesis Block فالبالتالي جميع مكوناته تبنى من الصفر والـ Hash الخاص به يكون مبني على Hash سابق عشوائي. كالمثال التالي:

```
String [] genesisTransactions = {"Sultan sent Feras 10 bitcoin","Feras sent 100 bitcoins to
Hamad"};
Block genesisBlock = new Block(0, genesisTransactions);
blockchain.add(genesisBlock);
```

ولغرض التبسيط في هذا المثال, صنعنا `genesisTransactions` وهي عباره عن مصفوفة نصوص (strings) تمثل عمليات عشوائيه (fake transactions) وبدورها تمثل العمليات اللتي تمت في الـGenesis Block . وقمنا بصناعة الـ Genesis Block وكما نلاحظ طالما انه ليس لديه Block سابق, فالبالتالي الـ Hash السابق سيكون عباره عن صفر (zero) وبأمكانك ان تضع قيمه عشوائيه اخرى, المهم بأن هذا الـ Block في حالة صناعة المحتوى الخاص به لا يرتبط بمحتوى Block اخر, وهذا يعني ان الـ Hash الخاص به يعتمد إعتماد كلي على طريقتك في صناعته. بعد صناعة الـ Block الأول نقوم بإضافته في *blockchain* اللذي صنعناه في البدايه. لوقمنا بطباعة محتوى الـ Hash الخاص بهذا الـ Block نقوم بالتالي:

```
System.out.println("Hash of genesis block:");
System.out.println(blockchain.get(0).getBlockHash());
```

المخرجات ستكون كالتالي:

```
Hash of genesis block:
-325325674
```

نلاحظ بأن الرقم -325325674 عباره عن قيمة الـ Hash الخاص بالـGenesis Block, ومثل ماشفنا في طريقة بناء الـ Hash لكل Block بأنه يعتمد على محتوى الـ Block نفسه (يمثله العمليات transactions اللي تمت في نفس Block), أي بمعنى انه اذا تغير اي شيء في الـ transactions سيؤثر على قيمة الـHash , مثلا:

```
String [] genesisTransactions = {"Sultan sent Feras 20 bitcoin","Feras sent 50 bitcoins to
Hamad"};
Block genesisBlock = new Block(0, genesisTransactions);
blockchain.add(genesisBlock);
```

قمنا بتغيير العمليتين التاليه: عوضا ان يرسل سلطان لفراس قيمه 10 بتكوين, اصبحت 20 بتكوين, وكذلك بنفس الشي بين فراس وحمد غيرنا القيمه من 100 إلى 50. لو قمنا بطباعة محتوى الـ Hash الخاص بالـ Genesis Block مره اخرى سيكون الناتج -729499183 مختلف عن القيمه السابقه, وهو دليل على تغير الـ Digital ID الخاص بهذا الـBlock. وهذا مهم جدا, لأنه عندما ننشئ الـ Block رقم 2 في السلسله, سيحتاج الـ Hash الخاص بالـ Block السابق (الـGenesis Block) ويحتاج ايضا نوع خاص فيه من الـTransactions. في الخطوه التاليه نوضح بالـ Code كيف نترجم هذا الكلام بلغة Java.

```
String [] block2Transactions = {"Ali sent 5 bitcoin to Sami", "Sultan received 2 bitcoin from
Ahmed"};
Block block2 = new Block(genesisBlock.getBlockHash(), block2Transactions);
blockchain.add(block2);
```

العمليات اللتي تمت في الـ Block رقم 2 هي كالتالي: علي ارسل 5 بيتكوين لسامي, و سلطان استقبل 2 بتكوين من أحمد, وهذي هيا الـTransactions الخاصه بالـ Block رقم 2 في هذا الـBlockchain. الأن بنفس الطريقه نقوم بطباعة الـDigital ID الخاص بالـ Block رقم 2:

```
System.out.println("Hash of block 2:");
System.out.println(blockchain.get(1).getBlockHash());
```

نلاحظ بأن الـ Digital ID الخاص بالـBlock رقم 2 هو 1908707073.الجميل في الموضوع هو لو قمنا فقط بتغيير محتوى العمليات (الـTransactions) في الـ Block رقم 1 (Genesis Block) مثلا عوضا ان يرسل سلطان لفراس 20 بتكوين, اصبحت 40 بتكوين, وقمنا بطباعة الـ Digital ID الخاص بالـ Blocks رقم 1 و 2 معا, ماذا سيحدث؟

```
String[] genesisTransactions = {"Sultan sent Feras 40 bitcoin","Feras sent 50 bitcoins to
Hamad"};
Block genesisBlock = new Block(0, genesisTransactions);
blockchain.add(genesisBlock);

String[] block2Transactions = {"Ali sent 5 bitcoin to Sami", "Sultan recived 2 bitcoin from
Ahmed"};
Block block2 = new Block(genesisBlock.getBlockHash(), block2Transactions);
blockchain.add(block2);
```

```
System.out.println("Hash of genesis block:");
System.out.println(blockchain.get(0).getBlockHash());

System.out.println("Hash of block 2:");
System.out.println(blockchain.get(1).getBlockHash());
```

سنلاحظ بأن الـHash لكل الـ Blocks في هذا الـBlockchain تغيرت تماما بسبب التغيير البسيط الحاصل في الـ Genesis Block:

```
Hash of genesis block:
-471333745
Hash of block 2:
-2128094785
```

وهذا هو الأساس المبني عليه تقنية الـBlockchain, بأن أي تغيير على Block سيتم التأثير على جميع الـBlocks في هذه السلسله. مثلا لو عملنا Block رقم 3 مع الأخذ بعين الإعتبار أن الـHash السابق هو الـHash الخاص بالـ Block رقم 2 والـ Transactions الخاصه بهذا الـ Block (رقم 3) هي فقط "أحمد ارسل 100 بتكوين لأمه" وقمنا بنفس العمليه السابقه (نغير محتويات الـGenesis Block فقط) سنلاحظ التغيير يأثر على جميع الـ Blocks (رقم 2 و 3) المرتبطه مع بعض في الـBlockchain.

```
void buildBlockchain(ArrayList<Block> blockchain){
	String[] genesisTransactions = { "Sultan sent Feras 40 bitcoin",
			"Feras sent 50 bitcoins to Hamad" };
	Block genesisBlock = new Block(0, genesisTransactions);
	blockchain.add(genesisBlock);

	String[] block2Transactions = { "Ali sent 5 bitcoin to Sami",
			"Sultan recived 2 bitcoin from Ahmed" };
	Block block2 = new Block(genesisBlock.getBlockHash(),block2Transactions);
	blockchain.add(block2);

	String[] block3Transactions = { "Ahamed sent 100 bitcoin to his mom" };
	Block block3 = new Block(block2.getBlockHash(), block3Transactions);
	blockchain.add(block3);

	System.out.println("Hash of genesis block:");
	System.out.println(blockchain.get(0).getBlockHash());

	System.out.println("Hash of block 2:");
	System.out.println(blockchain.get(1).getBlockHash());

	System.out.println("Hash of block 3:");
	System.out.println(blockchain.get(2).getBlockHash());
}
```

أخيرا, الـ Hashing Algorithms صعب جدا كسرها, بمعنى يصعب على المخترق ان يتعرف على طريقة عمل الخوارزميه اللتي تنتج الـHash وبالتالي يستطيع ان يخمن الأجوبه الصحيحه لكل Block. وبالتالي كلما زاد عدد الـ Blocks في السلسله (chain) كلما زادت عملية التعقيد لكسر الـ Hashes المرتبطه مع بعضها البعض, لأن كما ذكرت سابقا بان كل Hash مرتبط حسابيا بالـ Hash اللذي قبله ولذلك اذا اردنا كسر (Crack) الـ Hash الخاص بالـBlock رقم 3 يلزمك ان تقوم بكسر الـ Hash اللذي يعتمد (الـ Hash الخاص برقم 2) ولكي تنجح بكسر الـ Hash السابق يلزمك ان تعرف الـHash السابق وهكذا. وبالتالي هذه العمليه اللتي تجعل من تقنية الـBlockchain اكثر امان لتداول عملة الـBitcoin وعليه لو أن شخص في شبكة الـBlockchain عمل تغيير في سجلاته (Transactions) الجميع في الشبكه سيعرفون انه عمل تغيير في الـTransactions لأن الـHash القديم لن يكون مطابق للـHash الجديد, فبالتالي بناء على البروتوكول المتبع في الـBitcoin يجب التحقق من صحة الـHash الجديد هل هو عمليه صحيحه تمت ام لا, وعليه ندخل في مفهوم الـBitcoin mining ومفهوم الـ Ledger (مفهوم المحفظه الماليه) الخاصه بتعاملات الـBitcoin. سنحاول ان نفرد درس خاص بهذا الموضوع مستقبلا أن شاء الله.