```java
package blockchaintask;

import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

/**
 * class Block:
 * The cass block is used to create a single block in the blockchain
 * It is also used to compute the proof of work and calculate the hash of a particular block
 * The block created stores the hash of the previous block
 */
public class Block {
    // Private variable of the block that are required to identify a block.
    // Like its index, the timestamp is which it was created
    // The nonce of the block, the transaction data and the previous hash
    private int index, difficulty;
    private BigInteger nonce;
    private Timestamp timestamp;
    private String data, previousHash;

    /**
     * Block Constructor
     * The constructor is used to create a block and initialize it with the values provided from the chain
     * @param index the number of the block in the blockchain
     * @param timestamp the timestamp at which the block was created
     * @param data the transaction data at the block
     * @param difficulty the difficulty to calculate the hash of the block
     */
```

```
32      Block(int index, Timestamp timestamp, String data
   , int difficulty)
33      {
34          this.index = index;
35          this.timestamp = timestamp;
36          this.data = data;
37          this.difficulty = difficulty;
38      }
39
40      /**
41       * function setIndex
42       * Sets the index of the block
43       * @param index
44       */
45      public void setIndex(int index) {
46          this.index = index;
47      }
48
49      /**
50       * function setDifficulty
51       * Sets the difficulty of the block
52       * @param difficulty
53       */
54      public void setDifficulty(int difficulty) {
55          this.difficulty = difficulty;
56      }
57
58      /**
59       * function setTimestamp
60       * Sets the timestamp of the block
61       * @param timestamp
62       */
63      public void setTimestamp(Timestamp timestamp) {
64          this.timestamp = timestamp;
65      }
66
67      /**
68       * function setData
69       * Sets the data/transaction of the block
70       * @param data
71       */
```

```java
    public void setData(String data) {
        this.data = data;
    }

    /**
     * function setPreviousHash
     * set the hash of the block's parent
     * @param previousHash
     */
    public void setPreviousHash(String previousHash
) {
        this.previousHash = previousHash;
    }

    /**
     * function getIndex
     * gets the index of the block
     * @return index
     */
    public int getIndex() {
        return index;
    }

    /**
     * function getDifficuty
     * gets the difficulty of the block
     * @return difficulty
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * function getTimestamp
     * gets the timestamp of the block
     * @return timestamp
     */
    public Timestamp getTimestamp() {
        return timestamp;
    }

```

```
112     /**
113      * function getData
114      * gets the transaction of the block
115      * @return data
116      */
117     public String getData() {
118         return data;
119     }
120 
121     /**
122      * function getPreviousHash
123      * gets the parent's hash of the block
124      * @return previous hash
125      */
126     public String getPreviousHash() {
127         return previousHash;
128     }
129 
130     /**
131      * function getNonce
132      * gets the nonce of the block
133      * @return nonce
134      */
135     public BigInteger getNonce()
136     {
137         return nonce;
138     }
139 
140     /**
141      * function calculate hash
142      * calculates the hash of of a string
143      * @return hashed String
144      * @throws NoSuchAlgorithmException for the SHA
    calculation
145      */
146     public String calculateHash() throws
    NoSuchAlgorithmException {
147         //Our string contains the index, timestamp,
    data, previous hash, nonce and difficulty
148         String hash = index + timestamp.toString
    () + data + previousHash + nonce + difficulty;
```

```
149         MessageDigest digest;
150         //Getting the instance of SHA-256
151         digest = MessageDigest.getInstance("SHA-256"
    );
152         //encoding with SHA-256
153         byte[] encodedhash = digest.digest(
154                 hash.getBytes(StandardCharsets.UTF_8
    ));
155         return bytesToHex(encodedhash);
156     }
157 
158     /**
159      * function byteToHex
160      * The function is used to convert byte to hex
    of the encoded string
161      * @pargm hash takes the hash
162      * @return
163      */
164     private static String bytesToHex(byte[] hash) {
165         //The hexString is double the length of the
    hash
166         StringBuilder hexString = new StringBuilder(
    2 * hash.length);
167         //Convert each value of string to a hex
168         for (int i = 0; i < hash.length; i++) {
169             String hex = Integer.toHexString(0xff &
    hash[i]);
170             if(hex.length() == 1) {
171                 hexString.append('0');
172             }
173             //append the values together
174             hexString.append(hex);
175         }
176         //return the string
177         return hexString.toString().toUpperCase();
178     }
179 
180     /**
181      * function proofOfWork
182      * The function does the proof of work of a
    block
```

```
183      * It calculates the hash based on the
    difficulty and increases the  nonce till
184      * the initial 0s are not equal to the
    difficulty
185      * @return the hash value after the proof of
    work
186      * @throws NoSuchAlgorithmException
187      */
188     public String proofOfWork() throws
    NoSuchAlgorithmException {
189         //The nonce is initialized to 0
190         this.nonce = BigInteger.ZERO;
191         //calculate hash with 0 nonce
192         String h = calculateHash();
193         //string builder of number of 0s equal to
    the difficulty number
194         StringBuilder sb = new StringBuilder();
195         for(int i = 0; i < difficulty; i++)
196         {
197             sb.append("0");
198         }
199         //if the number of 0s in the stringbuilder
    match the start of the hash then we stop calculating
    , else we continue calculating
200         while(!(h.substring(0,difficulty).equals(sb.
    toString()))) {
201             this.nonce = nonce.add(BigInteger.ONE);
    //adding one to nonce everytime the string does not
    match
202             h = calculateHash(); //calculating hash
    again
203         }
204         //returning the hash
205         return h;
206     }
207
208     /**
209      * function toString
210      * The function is used to display the data of
    the blocks
211      * @return String value
```

```
212      */
213     @Override
214     public String toString() {
215         return "{" +
216                 "\"index\" : \"" + index + "\"" +
217                 ",\"time stamp\" : \"" + timestamp
    + "\"" +
218                 ", \"Tx\" : \"" + data + "\"" +
219                 ", \"PrevHash\" : \"" + previousHash
    + "\"" +
220                 ", \"nonce\" : \"" + nonce + "\"" +
221                 ", \"difficulty\" : \"" + difficulty
    + "\"" +
222                 "}";
223     }
224 }
225
```