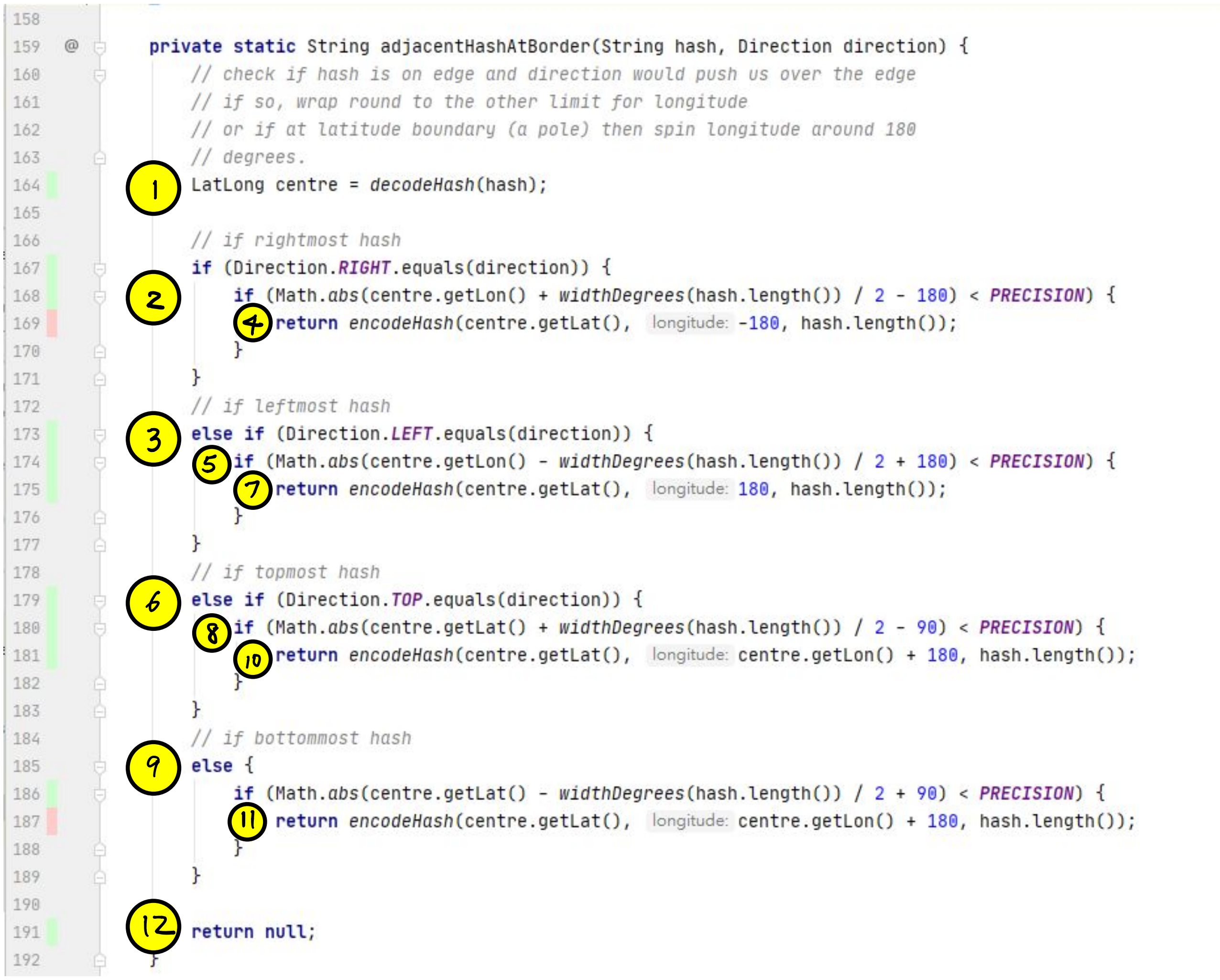
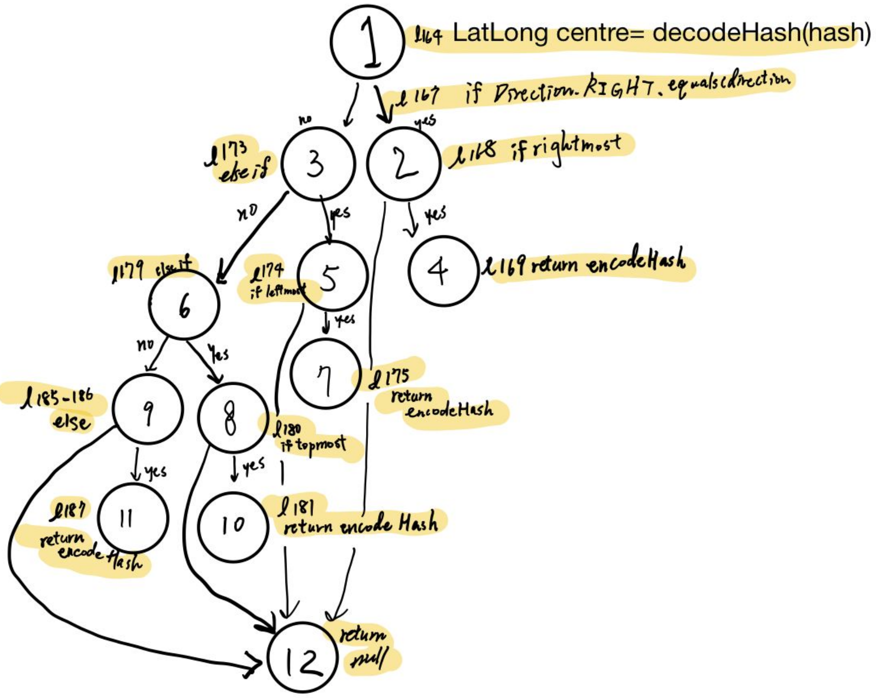
GeoHash.java

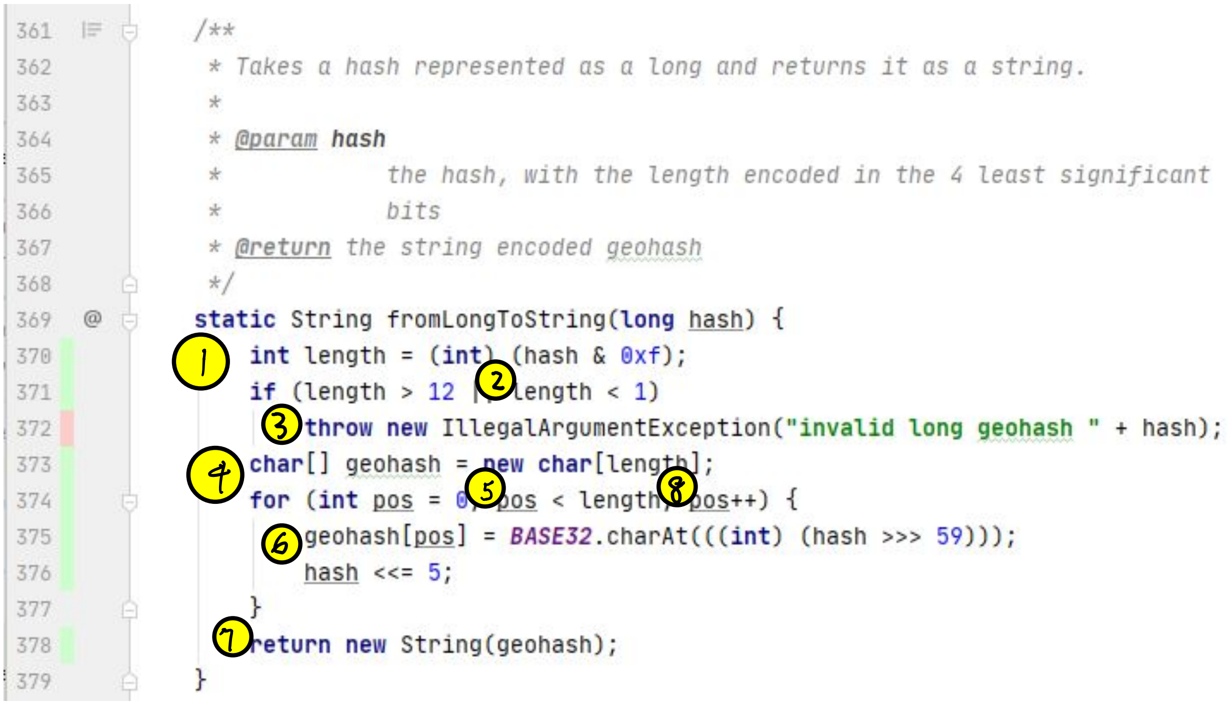
1. adjacentHashAtBorder(String hash, Direction direction)



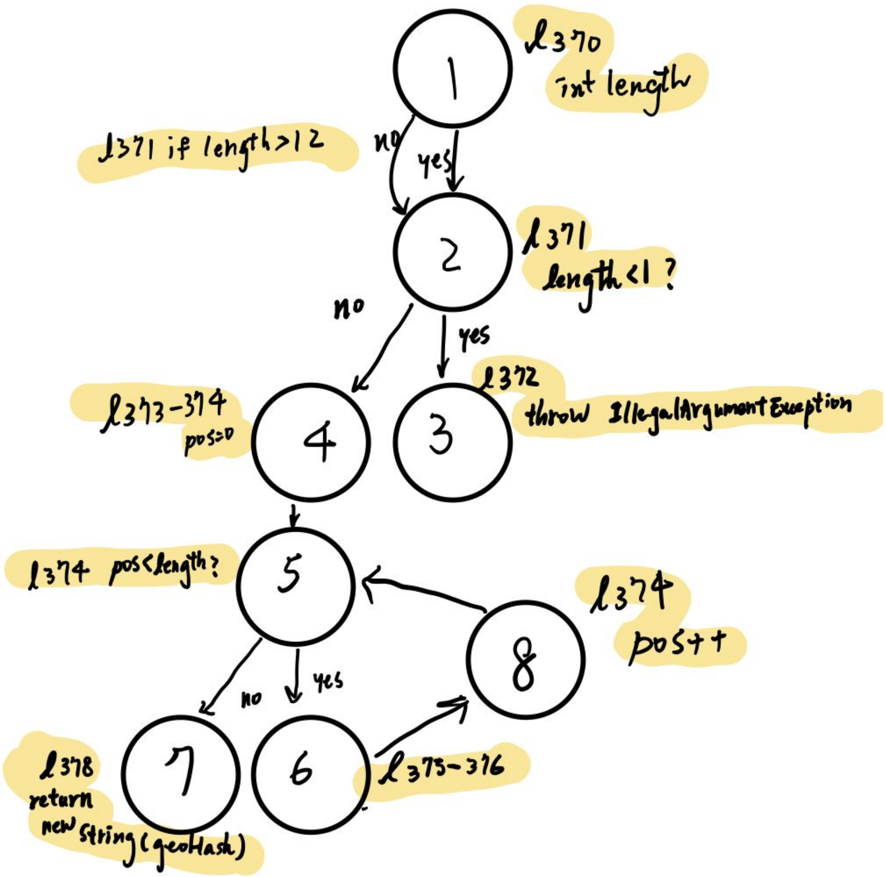
```
158
159 @     private static String adjacentHashAtBorder(String hash, Direction direction) {
160           // check if hash is on edge and direction would push us over the edge
161           // if so, wrap round to the other limit for longitude
162           // or if at latitude boundary (a pole) then spin longitude around 180
163           // degrees.
164           LatLong centre = decodeHash(hash);
165
166           // if rightmost hash
167           if (Direction.RIGHT.equals(direction)) {
168               if (Math.abs(centre.getLon() + widthDegrees(hash.length()) / 2 - 180) < PRECISION) {
169                   return encodeHash(centre.getLat(),  longitude: -180, hash.length());
170               }
171           }
172           // if leftmost hash
173           else if (Direction.LEFT.equals(direction)) {
174               if (Math.abs(centre.getLon() - widthDegrees(hash.length()) / 2 + 180) < PRECISION) {
175                   return encodeHash(centre.getLat(),  longitude: 180, hash.length());
176               }
177           }
178           // if topmost hash
179           else if (Direction.TOP.equals(direction)) {
180               if (Math.abs(centre.getLat() + widthDegrees(hash.length()) / 2 - 90) < PRECISION) {
181                   return encodeHash(centre.getLat(),  longitude: centre.getLon() + 180, hash.length());
182               }
183           }
184           // if bottommost hash
185           else {
186               if (Math.abs(centre.getLat() - widthDegrees(hash.length()) / 2 + 90) < PRECISION) {
187                   return encodeHash(centre.getLat(),  longitude: centre.getLon() + 180, hash.length());
188               }
189           }
190
191           return null;
192       }
```



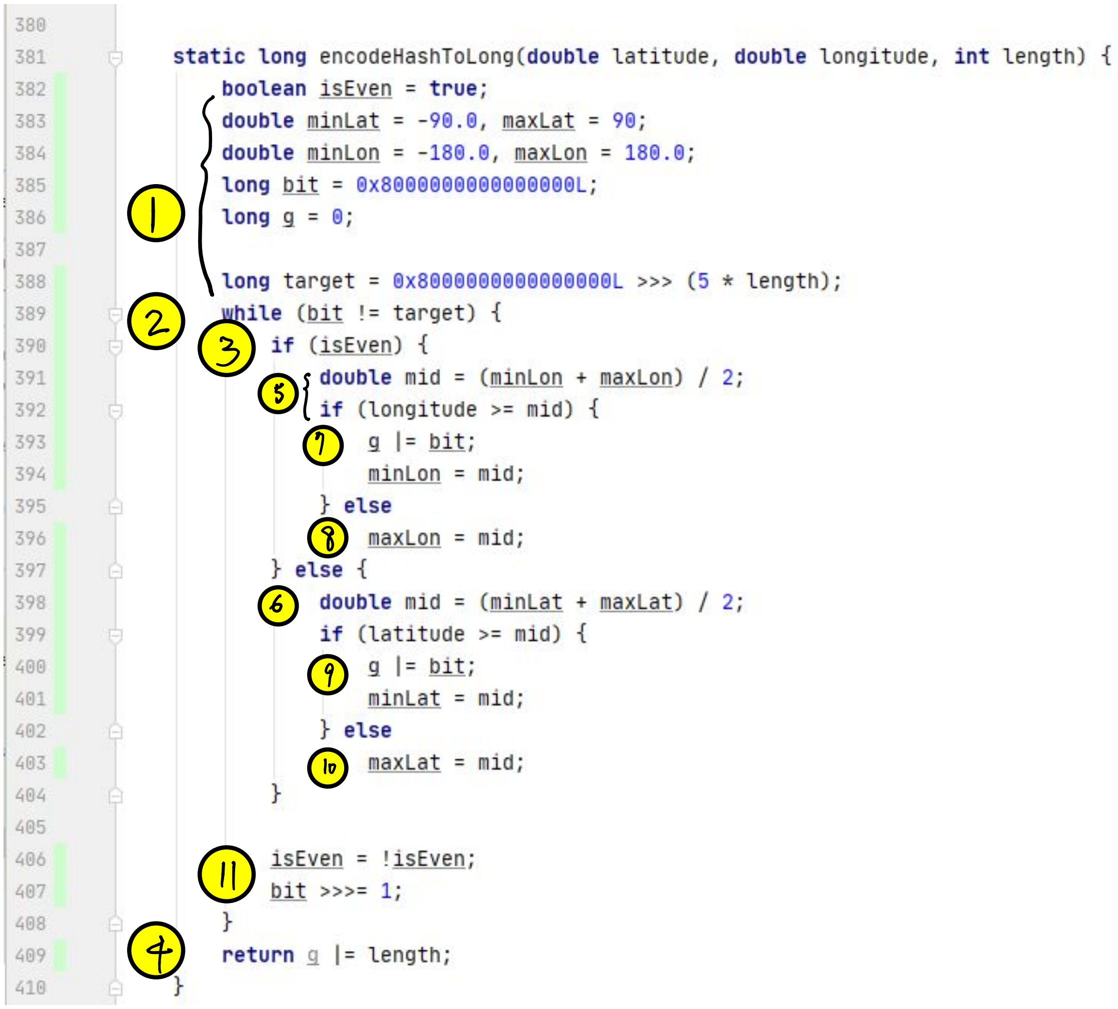
2. fromLongToString(long hash)



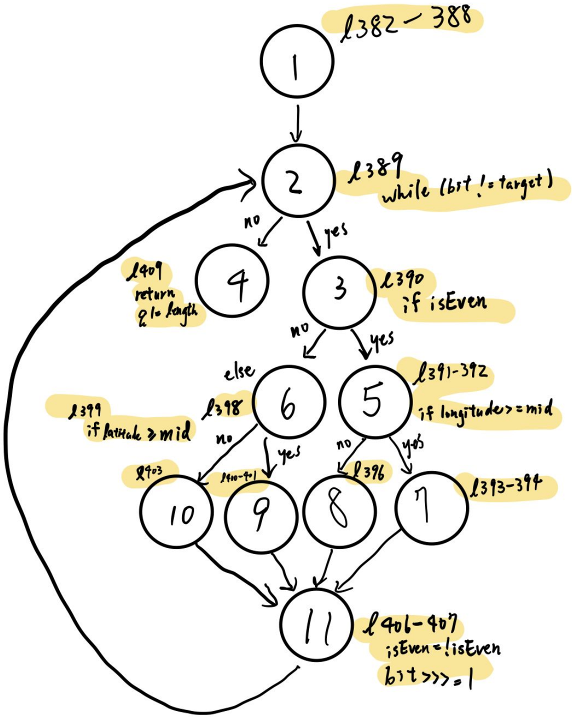
```
361 /**
362  * Takes a hash represented as a long and returns it as a string.
363  *
364  * @param hash
365  *            the hash, with the length encoded in the 4 least significant
366  *            bits
367  * @return the string encoded geohash
368  */
369 static String fromLongToString(long hash) {
370     int length = (int) (hash & 0xf);
371     if (length > 12 || length < 1)
372         throw new IllegalArgumentException("invalid long geohash " + hash);
373     char[] geohash = new char[length];
374     for (int pos = 0; pos < length; pos++) {
375         geohash[pos] = BASE32.charAt(((int) (hash >>> 59)));
376         hash <<= 5;
377     }
378     return new String(geohash);
379 }
```



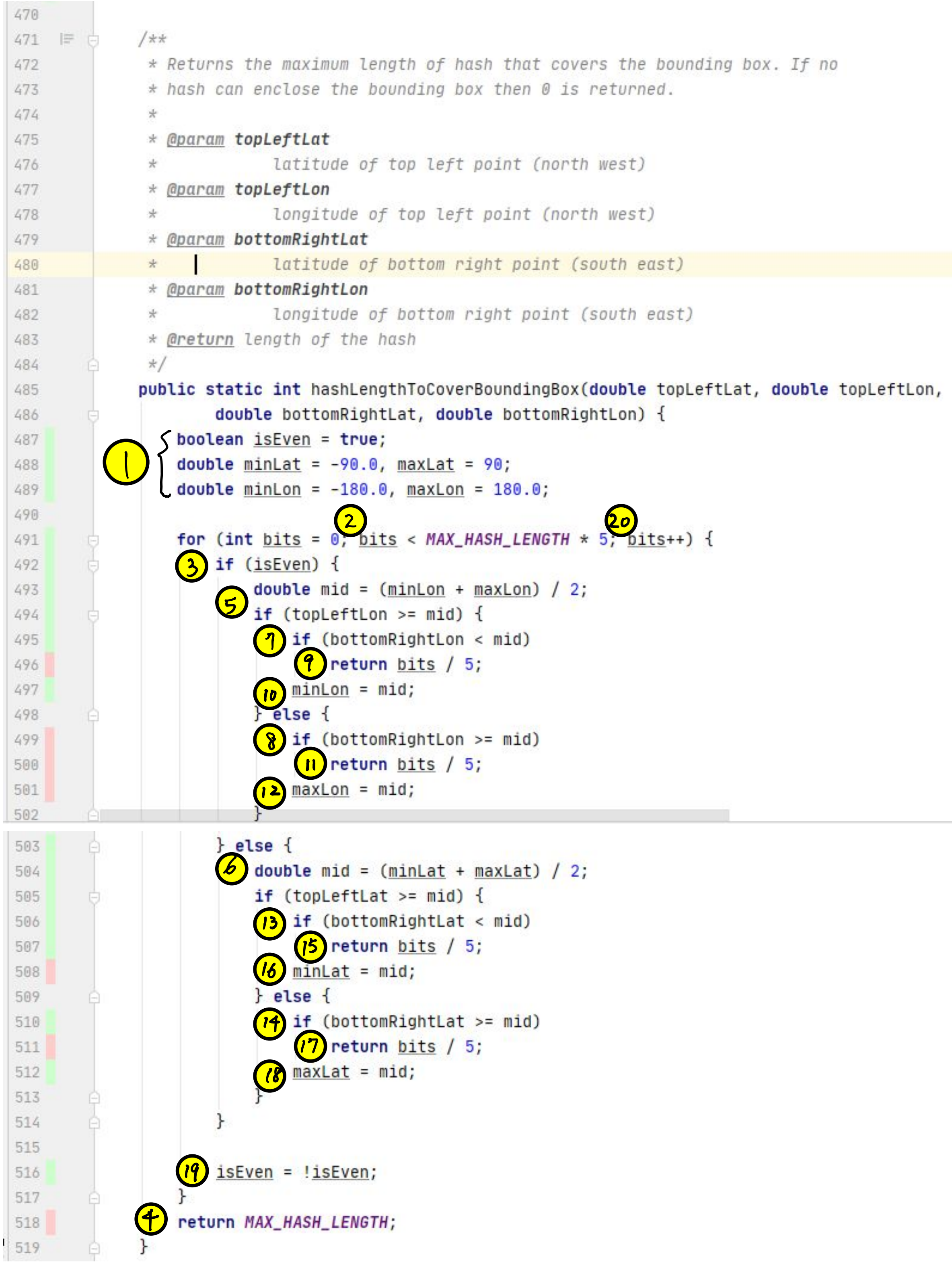
3. encodeHashToLong(double latitude, double longitude, int length)



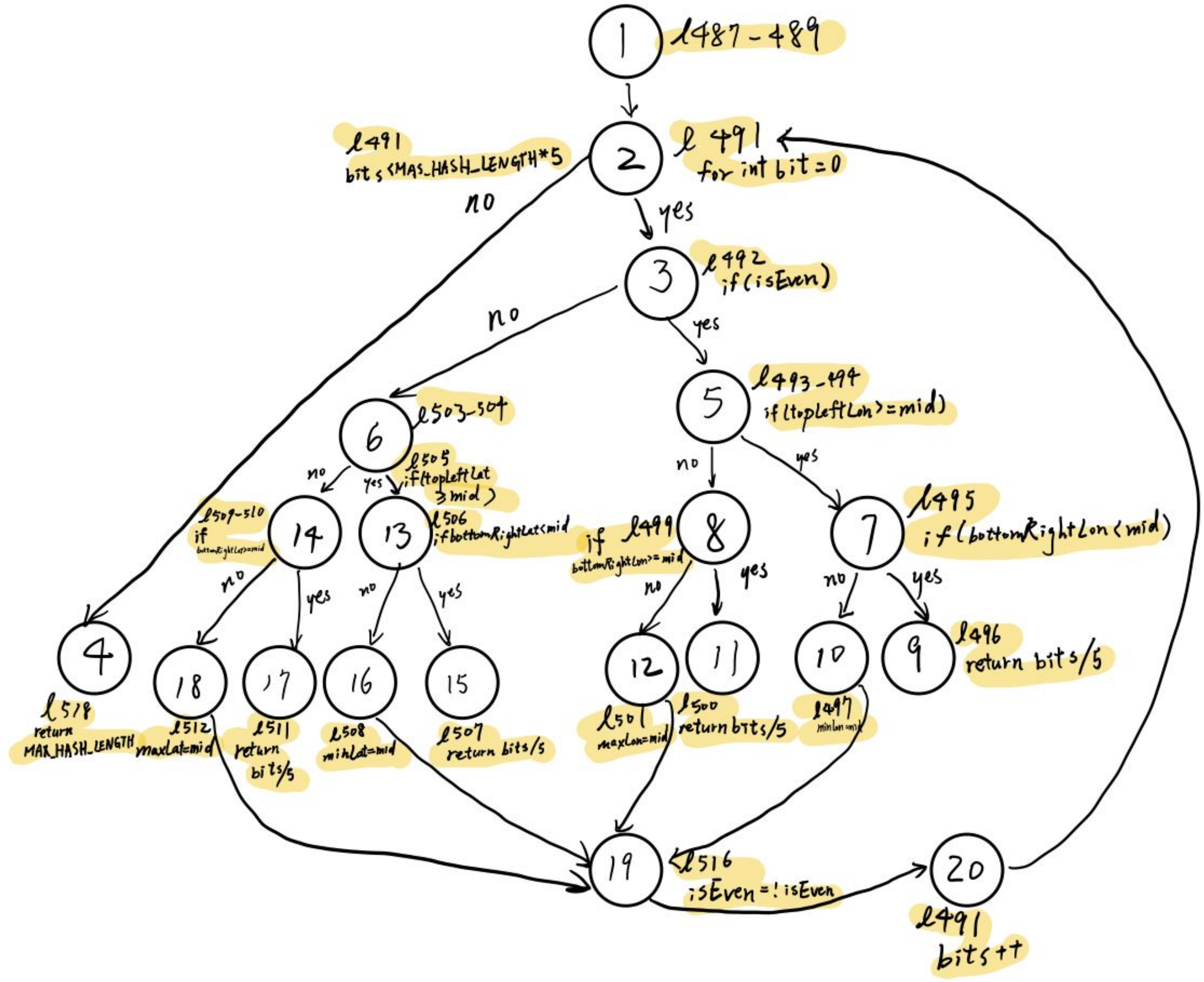
```
380
381     static long encodeHashToLong(double latitude, double longitude, int length) {
382         boolean isEven = true;
383         double minLat = -90.0, maxLat = 90;
384         double minLon = -180.0, maxLon = 180.0;
385         long bit = 0x8000000000000000L;
386         long g = 0;
387
388         long target = 0x8000000000000000L >>> (5 * length);
389         while (bit != target) {
390             if (isEven) {
391                 double mid = (minLon + maxLon) / 2;
392                 if (longitude >= mid) {
393                     g |= bit;
394                     minLon = mid;
395                 } else
396                     maxLon = mid;
397             } else {
398                 double mid = (minLat + maxLat) / 2;
399                 if (latitude >= mid) {
400                     g |= bit;
401                     minLat = mid;
402                 } else
403                     maxLat = mid;
404             }
405
406             isEven = !isEven;
407             bit >>>= 1;
408         }
409         return g |= length;
410     }
```



4. hashLengthToCoverBoundingBox(double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon)

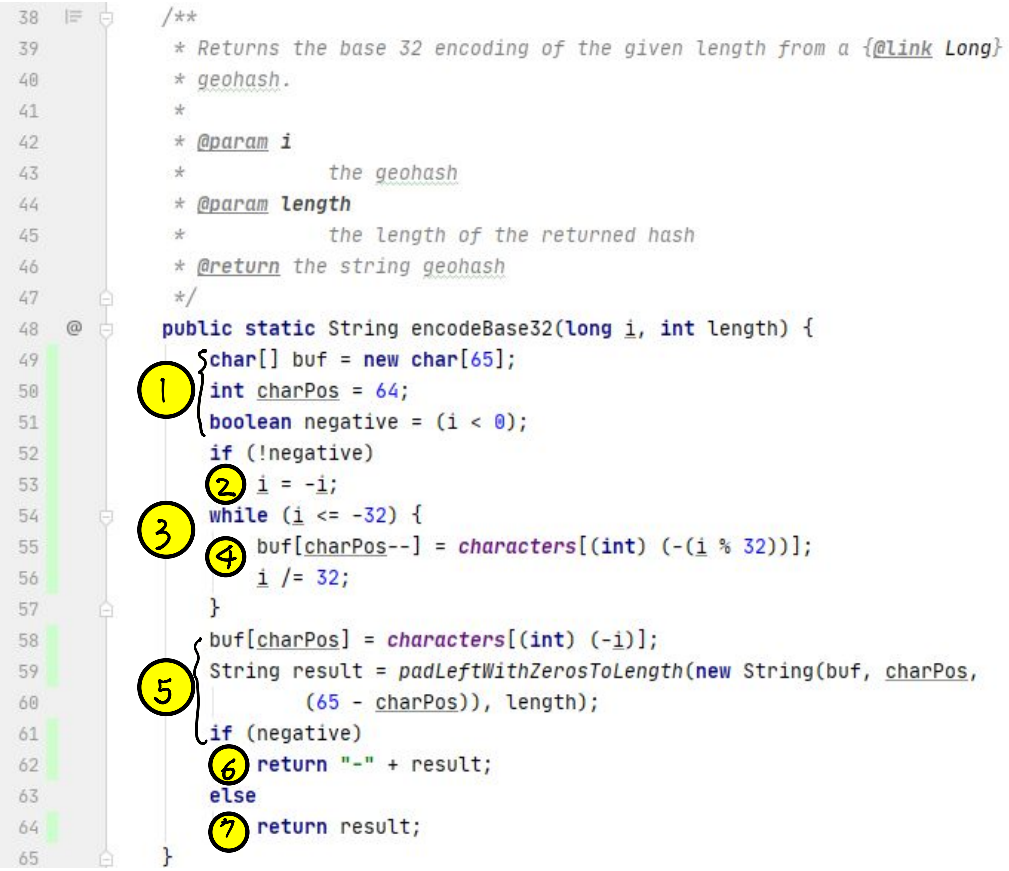


```java

    /**
     * Returns the maximum length of hash that covers the bounding box. If no
     * hash can enclose the bounding box then 0 is returned.
     *
     * @param topLeftLat
     *            latitude of top left point (north west)
     * @param topLeftLon
     *            longitude of top left point (north west)
     * @param bottomRightLat
     *            latitude of bottom right point (south east)
     * @param bottomRightLon
     *            longitude of bottom right point (south east)
     * @return length of the hash
     */
    public static int hashLengthToCoverBoundingBox(double topLeftLat, double topLeftLon,
            double bottomRightLat, double bottomRightLon) {
        boolean isEven = true;
        double minLat = -90.0, maxLat = 90;
        double minLon = -180.0, maxLon = 180.0;

        for (int bits = 0; bits < MAX_HASH_LENGTH * 5; bits++) {
            if (isEven) {
                double mid = (minLon + maxLon) / 2;
                if (topLeftLon >= mid) {
                    if (bottomRightLon < mid)
                        return bits / 5;
                    minLon = mid;
                } else {
                    if (bottomRightLon >= mid)
                        return bits / 5;
                    maxLon = mid;
                }
            } else {
                double mid = (minLat + maxLat) / 2;
                if (topLeftLat >= mid) {
                    if (bottomRightLat < mid)
                        return bits / 5;
                    minLat = mid;
                } else {
                    if (bottomRightLat >= mid)
                        return bits / 5;
                    maxLat = mid;
                }
            }

            isEven = !isEven;
        }
        return MAX_HASH_LENGTH;
    }
```

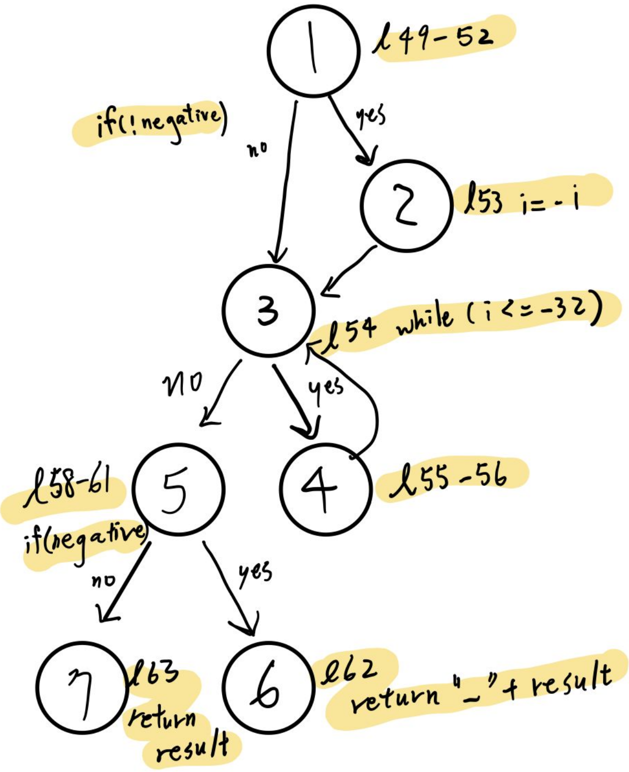

1
l487-489
2
l491
bits<MAS_HASH_LENGTH*5
l 491
for int bit=0
no
yes
3
l492
if(isEven)
no
yes
5
l493-494
if(topleftLon>=mid)
6
l503-504
l505
if(topleftLat
>=mid)
no
yes
no
yes
14
l509-510
if
bottomRightLat>=mid
13
l506
ifbottomRightLat<mid
8
if l499
bottomRightLon>=mid
7
l495
if(bottomRightLon<mid)
no
yes
no
yes
no
yes
no
yes
4
l518
return
MAR_HASH_LENGTH
18
l512
maxLat=mid
17
l511
return
bits/5
16
l508
minLat=mid
15
l507
return bits/5
12
l501
maxLon=mid
11
l500
return bits/5
10
l497
minLon=mid
9
l496
return bits/5
19
l516
isEven=!isEven
20
l491
bits++

Base32

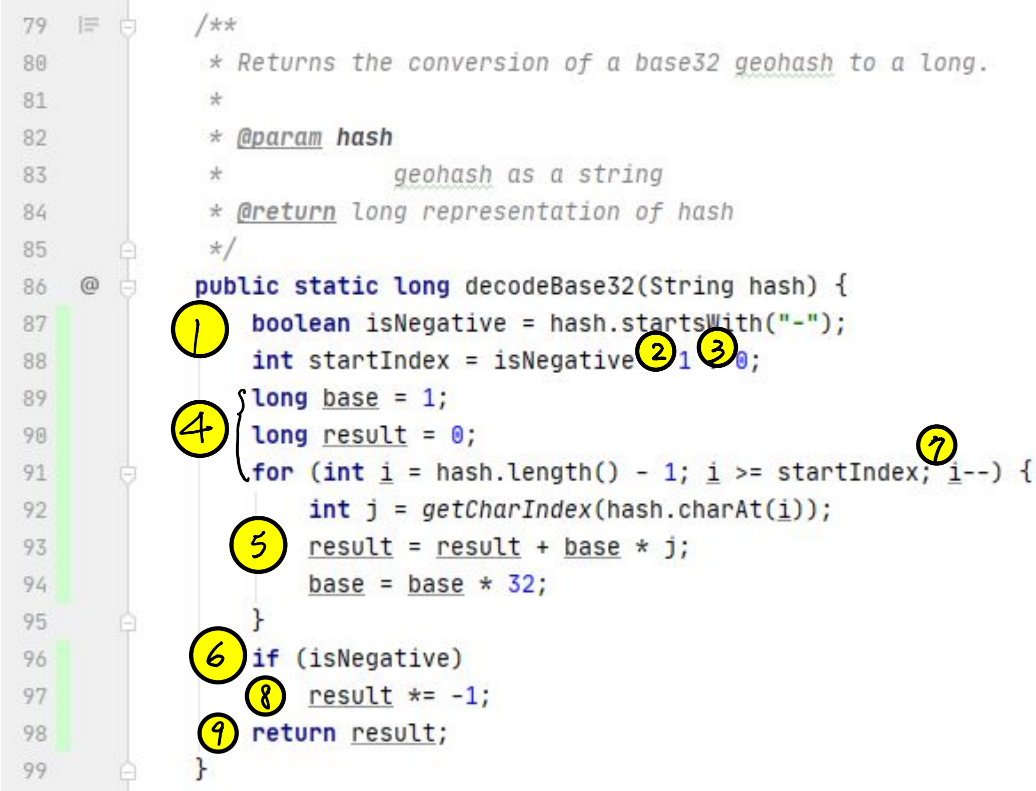
5. encodeBase32(long I, int length)



```
38  /**
39   * Returns the base 32 encoding of the given length from a {@link Long}
40   * geohash.
41   *
42   * @param i
43   *            the geohash
44   * @param length
45   *            the length of the returned hash
46   * @return the string geohash
47   */
48  public static String encodeBase32(long i, int length) {
49      char[] buf = new char[65];
50      int charPos = 64;
51      boolean negative = (i < 0);
52      if (!negative)
53          i = -i;
54      while (i <= -32) {
55          buf[charPos--] = characters[(int) (-(i % 32))];
56          i /= 32;
57      }
58      buf[charPos] = characters[(int) (-i)];
59      String result = padLeftWithZerosToLength(new String(buf, charPos,
60              (65 - charPos)), length);
61      if (negative)
62          return "-" + result;
63      else
64          return result;
65  }
```

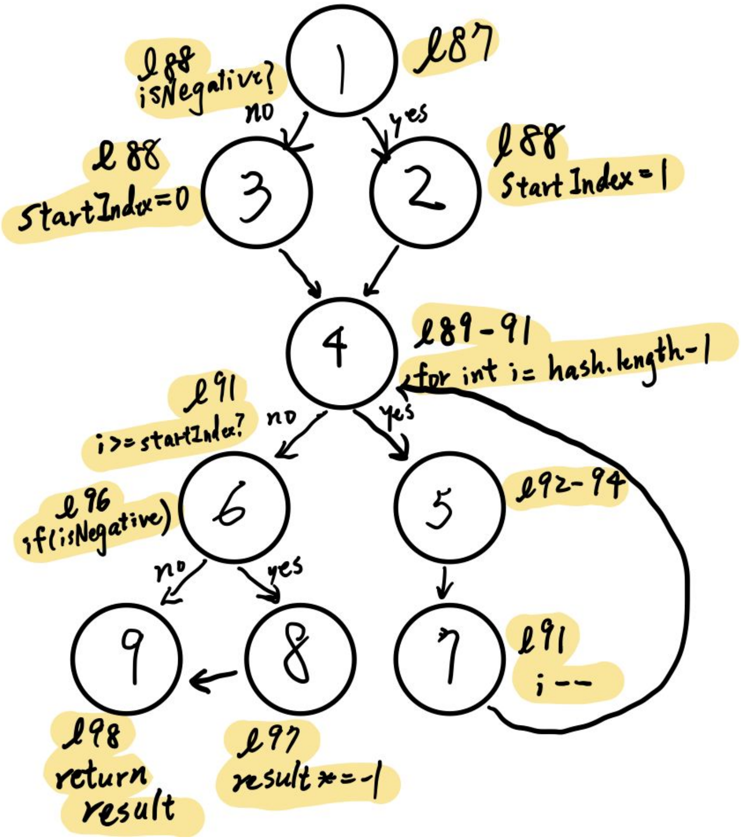


6. decodeBase32(String hash)



```
79      /**
80       * Returns the conversion of a base32 geohash to a long.
81       *
82       * @param hash
83       *            geohash as a string
84       * @return long representation of hash
85       */
86      public static long decodeBase32(String hash) {
87          boolean isNegative = hash.startsWith("-");
88          int startIndex = isNegative ? 1 : 0;
89          long base = 1;
90          long result = 0;
91          for (int i = hash.length() - 1; i >= startIndex; i--) {
92              int j = getCharIndex(hash.charAt(i));
93              result = result + base * j;
94              base = base * 32;
95          }
96          if (isNegative)
97              result *= -1;
98          return result;
99      }
```

1
2
3
4
7
5
6
8
9



l87
1
l88
isNegative?
no
yes
3
2
l88
StartIndex=0
l88
Start Index=1
4
l89-91
for int i= hash.length-1
l91
i>=startIndex?
no
yes
6
5
l96
if(isNegative)
l92-94
no
yes
9
8
7
l91
i--
l98
return
result
l97
result*=-1