Questionnaire of different bytecode-based comment genaration model

This questionnaire is aimed to investigate the effectiveness of several models in generating code comments under bytecode training. We used six anonymous models to generate comments.

The generated comments are evaluated from two aspects: naturalness and informativeness. Naturalness describes whether the generated comments are fluent and grammatically accurate; informativenessdescribes whether the generated comments correctly reflect the bytecode information.

We provide the source code as a reference, that is, it is necessary to judge whether the comment correctly reflects the function of the source code. Both naturalness and informativeness are scored on a scale of 0-4:
A. Naturalness
0: The sentence is incomplete, illogical and has grammatical errors.
1: The sentence is basically complete, but not smooth and has grammatical errors.
2: The sentence is relatively complete, basically smooth but with grammatical errors.
3: The sentence is relatively complete, basically smooth and without grammatical errors.
4: Sentences are very complete, smooth and free from grammatical
B.Informativeness
0: Comment error describing the source code function.
1: There are a few descriptions in the comments that correctly reflect the function of the source code, but most of the descriptions are wrong.
2: Most of the descriptions of the comments correctly reflect the source code functions, and a small part of the descriptions are wrong.
3: Comments can basically and accurately describe the function of the source code.
4: Comments can very accurately describe the function of the source code.

1. Your basic information

Name	_____
Age	_____
Affiliated institution	_____

Programming experience(years)	_____
Programming languages you are at (e.g., Java, C++, Python etc.)	_____
English level （e.g., CET4, CET6, IELTS, TOEFL etc. ）	_____

2.

```
1 public void setAttribute(StackMap sm) {
2     AttributeInfo.remove(attributes, StackMap.tag);
3     if (sm != null) attributes.add(sm);
4 }
```

reference: adds a stack map table for j2me cldc

comment1: adds a stack map table for j2me cldc

comment2: adds the query attribute

comment3: adds a stack map

comment4: sets the list of attribute for the stack

comment5: sets a stack map of attribute

comment6: set a new attribute

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

3.

```
public String[] getDeclaredInterfaceNames() {
    int[] indexes = getDeclaredInterfaceIndexes();
    String[] names = new String[indexes.length];
    ClassEntry entry; for (int i = 0; i < indexes.length; i++) {
        entry = (ClassEntry) getPool().getEntry(indexes[i]);
        names[i] = _project.getNameCache().getExternalForm (entry.getNameEntry().getValue(), false);
    }
    return names;
}

```

reference: return the name of the interface declare for this class <SEG> include package name <SEG> or an empty array if none

comment1: return the name of all the interface this class <seg> include those of that class

comment2: return the name of the interface declare for this class <seg> include package name <seg> or an empty array if none

comment3: return the name of all unique interface

comment4: return the name of the interface declare in this class

comment5: get the name of all interface

comment6: returns the name of the name of the name of name of the field

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○

Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

4.

```
public static String methodDescriptorWithoutReturnType(MethodNode mNode) {
    StringBuilder sb = new StringBuilder();
    sb.append(mNode.getName()).append(':');
    for (Parameter p : mNode.getParameters()) {
        sb.append(ClassNodeUtils.formatTypeName(p.getType())).append(',');
    }
    return sb.toString();
}

```

reference: return the method node 's descriptor include it name and parameter type without generic

comment1: return type name of it return type next exception type which include it return

comment2: return the method node 's descriptor include it name and parameter type

comment3: return the method node 's descriptor

comment4: returns the type descriptor

comment5: returns a descriptor for a method

comment6: returns a descriptor for a method

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○

Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

5.

```
public RangeInfo subListBorders(int size) {
    if (inclusive == null) {
        throw new IllegalStateException("Should not call subListBorders on a non-inclusive aware IntRange"); }
    return subListBorders(from, to, inclusive, size);
}

```

reference: a method for determine from and to information when use this intrange to index an aggregate object of the specify size

comment1: a method for determine from and to information when use this intrange to index an aggregate object of the specify size

comment2: a method for determine from and to information

comment3: returns a substring from the give range

comment4: returns a substring from the give string

comment5: returns a sub list of the sub list of sublist

comment6: returns an order - - - - - - - - - - - - - - - - - -

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○

Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

6.

```
public static String getPropNameForAccessor(String accessorName) {
    if (!isValidAccessorName(accessorName))
        return accessorName;
    int prefixLength = accessorName.startsWith("is") ? 2 : 3;
    return String.valueOf(accessorName.charAt(prefixLength)).toLowerCase() + accessorName.substring(prefixLength + 1);
}

```

reference: sets the current childpropertysetter < br > it will assign defaultchildpropertysetter if null < br > it accept a childpropertysetter instance or a closure

comment1: set the property of the current distinguish property < br > it will assign a a give instance to a it current position

comment2: sets the current childpropertysetter < br > it will assign defaultchildpropertysetter if null < br > it accept a childpropertysetter instance or a closure

comment3: set the property of the current distinguish property < br > it will assign a a give instance to a it current position

comment4: sets the child of the child

comment5: sets the parent of this builder

comment6: sets the child property setter

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

7.

```
public void setNextContinuationToken(String nextContinuationToken) {
    this.nextContinuationToken = nextContinuationToken;
}

```

reference: sets the optional nextcontinuationtoken

comment1: sets the optional nextcontinuationtoken

comment2: sets the optional continuation token

comment3: sets the next continuation token

comment4: return the next token

comment5: sets next token

comment6: sets the next token

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○

Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

8.

```
1 public TreeData updateNodeTracker(final NodeTracker newTracker) {
2     return new TreeData(root, parentMapping, replacementMapping, newTracker,
  referenceTracker);
3 }
4
```

reference: creates a new instance which use the specified { code nodetracker }

comment1: creates a new instance which use the specified { code referencetracker }

comment2: creates a new instance which use the specified { code nodetracker }

comment3: creates a new instance of { link nodetracker } param { code nodetracker } return new { code treetrategy }

comment4: returns a { code treetracker } with the same mapping a the specify { code nodetracker }

comment5: < p > creates a new { link nodelist } that be the root node

comment6: creates a new { link treenode } instance with the give node

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○

Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

9.

```
public long readLong() {
    // Skip leading spaces skipSpaces();
    int start = index;
    while (index < size && Character.isDigit((char)buffer[index]))
        index++;
    if (index > start) {
        try {
            return ASCIIUtility.parseLong(buffer, start, index);
        } catch (NumberFormatException nex) { }
    }
    return -1;
}

```

reference: extract a long number <SEG> start at the current position

comment1: reads the primary bit <seg> exclude the give object

comment2: extract the < code > long < code > from the give long

comment3: reads a value from the buffer

comment4: reads a long integer value from the buffer

comment5: reads the next byte from this input stream

comment6: skips the specified number of byte from the give offset

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○

Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

10.

```
public Quota[] getQuota() throws MessagingException {
    return (Quota[])doOptionalCommand("QUOTA not supported",
            new ProtocolCommand() {
                public Object doCommand(IMAPProtocol p)
                throws ProtocolException { return p.getQuotaRoot(fullName); }
            });
}

```

reference: get the quota for the quotaroot associate with this folder

comment1: get the quota for the quotaroot associate with this folder

comment2: constructs a new subscribe that match the query associate with this folder

comment3: return the quota associate with the quota

comment4: get the quota for this folder

comment5: returns an array of all the quota

comment6: get the quota quota for the quota

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○

Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

11.

```
1 public IRubyObject write_long_long(ThreadContext context, IRubyObject value) {
2     getMemoryIO().putLong(0, Util.int64Value(value));
3     return this;
4 }
5
```

reference: writes a 64 bit integer value to the memory area

comment1: writes a 64 bit integer value to the memory area

comment2: writes a 64 bit integer value to the memory area

comment3: writes an unsigned integer value to the memory area

comment4: writes a 64 bit integer value to the memory area

comment5: writes a value to the memory

comment6: writes a long value from the object

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○

Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

12.

```
1 public int getCalculatedNumToSelect() {
2     if (m_numToSelect >= 0) {
3         m_calculatedNumToSelect = m_numToSelect > m_attributeMerit.length ? m_attributeMerit.length : m_numToSelect;
4     }
5     return m_calculatedNumToSelect;
6 }
7
```

reference: gets the calculated number to select

comment1: gets the calculated number of attribute to retain

comment2: performs a description of attribute

comment3: calculates the number of attribute select

comment4: gets the calculated number of attribute to select

comment5: returns the number of byte that have be select

comment6: returns the number of attribute in this map

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

13.

```
1 public int getNumFolds() {
2     return m_NumXValFolds;
3 }
4
```

reference: gets the number of fold for cross-validation

comment1: gets the number of fold for the cross-validation

comment2: sets the conditional of the attribute

comment3: returns the number of fold for the line

comment4: returns the number of fold for the cross - validation

comment5: returns the number of fold to the current number of fold

comment6: returns the number of fold the fold

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○

Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

14.

```
1  public BallNode buildTree() throws Exception {
2      ArrayList<TempNode> list = new ArrayList<TempNode>();
3      for(int i=0; i<m_InstList.length; i++) {
4          TempNode n A= new TempNode();
5          n.points = new int[1];
6          n.points[0] = m_InstList[i];
7          n.anchor = m_Instances.instance(m_InstList[i]);
8          n.radius = 0.0; list.add(n);
9      }
10     return mergeNodes(list, 0, m_InstList.length-1, m_InstList);
11 }
12
```

reference: builds the ball tree bottom up

comment1: builds a ball tree middle out

comment2: builds the ball tree bottom up

comment3: build the ball tree

comment4: builds a ball node

comment5: constructs a tree

comment6: builds a new node

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○

Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

15.

```
1 public OutputStream createBinaryFile(String typename) throws IOException {
2     if (verbose)
3         System.err.println("created binary: " + typename);
4         // KHK: for now the typename will already be a relative filename for the binary
5     String filename = typename.replace('.', File.separatorChar) + ".xsb";
6     File source = new File(classdir, typename); source.getParentFile().mkdirs();
7     return new FileOutputStream( source );
8 }
9
```

reference: creates a new schema binary file xsb and return a stream for write to it

comment1: creates a new output stream for write method

comment2: creates a new schema binary file xsb and return a stream for write with the content of the stream

comment3: creates a binary file

comment4: create a binary file

comment5: creates a binary file for the give file

comment6: return a steam for write to write

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○

Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

16.

```
public void putAll(Map<? extends String, ? extends GroovyRunner> m) {
    Map<String, GroovyRunner> map = getMap();
    writeLock.lock();
    try {
        cachedValues = null;
        for (Map.Entry<? extends String, ? extends GroovyRunner> entry : m.entrySet()) {
            if (entry.getKey() != null && entry.getValue() != null) {
                map.put(entry.getKey(), entry.getValue());
            }
        }
    }
    finally {
        writeLock.unlock();
    }
}

```

reference: adds all entry from the give map to the registry

comment1: set a runner runner with the specify key

comment2: adds all entry from the give map to the registry

comment3: adds all entry from the give map to the registry

comment4: puts all entry in the map

comment5: adds all of the mapping in this map

comment6: adds all of the entry in this set

	0	1	2	3	4
Naturalness					

Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

17.

```
1 public void addAllOf(AbstractByteList other) {
2     addAllOfFromTo(other, 0, other.size() - 1);
3 }
4
```

reference: appends all element of the specified list to the receiver

comment1: appends all of the element of the specified list to the receiver

comment2: inserts the specified list into the receiver

comment3: adds all element in the list

comment4: adds all the element element of the list

comment5: adds all the element in this list

comment6: adds all the element to the specified list

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○

Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

18.

```
static Formatter newFormatter(String name) throws Exception {
    return (Formatter) newObjectFrom(name, Formatter.class);
}

```

reference: creates a new formatter from the give class name

comment1: creates a new formatter from the give class name

comment2: creates a new formatter from the give class name

comment3: creates a new formatter for the give name

comment4: creates a new formatter

comment5: returns a formatter

comment6: returns a new log loggerter for the give string

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○

Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

19.

```
public static int getCodePointAt(Slice utf8, int position) {
    int unsignedStartByte = utf8.getByte(position) & 0xFF;
    if (unsignedStartByte < 0x80) {
        // normal ASCII // 0xxx_xxxx
        return unsignedStartByte;
    }
    if (unsignedStartByte < 0xc0) {
        // illegal bytes // 10xx_xxxx
        throw new InvalidUtf8Exception("Illegal start 0x" + toHexString(unsignedStartByte).to
    }
    if (unsignedStartByte < 0xe0) {
        // 110x_xxxx 10xx_xxxx
        if (position + 1 >= utf8.length()) {
            throw new InvalidUtf8Exception("UTF-8 sequence truncated");
        }
        return ((unsignedStartByte & 0b0001_1111) << 6) | (utf8.getByte(position + 1) & 0b001
    }
    if (unsignedStartByte < 0xf0) {
        // 1110_xxxx 10xx_xxxx 10xx_xxxx
        if (position + 2 >= utf8.length()) {
            throw new InvalidUtf8Exception("UTF-8 sequence truncated");
        }
        return ((unsignedStartByte & 0b0000_1111) << 12) | ((utf8.getByteUnchecked(position +
    }
    if (unsignedStartByte < 0xf8) {
        // 1111_0xxx 10xx_xxxx 10xx_xxxx 10xx_xxxx
        if (position + 3 >= utf8.length()) {
            throw new InvalidUtf8Exception("UTF-8 sequence truncated");
        }
        return ((unsignedStartByte & 0b0000_0111) << 18) | ((utf8.getByteUnchecked(position +
    }
    // Per RFC3629, UTF-8 is limited to 4 bytes, so more bytes are illegal
    throw new InvalidUtf8Exception("Illegal start 0x" + toHexString(unsignedStartByte).toUppe
}

```

reference: gets the utf-8 encode code point at the { code position }

comment1: gets the utf-8 sequence number for the current position

comment2: gets the utf-8 encode code point at the { code position }

comment3: gets the code point at the { code position }

comment4: returns the code point at the give index

comment5: gets the utf - 8 encode code point at the end

comment6: returns the code point code point of the specify code point of the give { code point }

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○

Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

20.

```java
public int getFeatureCount(Set<L> labels, double threshold, boolean useMagnitude) {
    if (labels != null) {
        Set<Integer> iLabels = getLabelIndices(labels);
        return getFeatureCountLabelIndices(iLabels, threshold, useMagnitude);
    }
    else {
        return getFeatureCount(threshold, useMagnitude);
    }
}

```

reference: returns number of feature with weight above a certain threshold

comment1: returns number of feature with weight of a certain threshold

comment2: returns number of feature with weight above a give profile

comment3: returns the number of feature with a give set

comment4: returns the number of feature in the list

comment5: returns the number of feature that have be add

comment6: returns < code > true < code > true < code > if < code >

	0	1	2	3	4
Naturalness					
Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○

21.

```
1 public static <E> Counter<E> division(Counter<E> c1, Counter<E> c2) {
2     Counter<E> result = c1.getFactory().create();
3     for (E key : Sets.union(c1.keySet(), c2.keySet())) {
4         result.setCount(key, c1.getCount(key) / c2.getCount(key));
5     }
6     return result;
7 }
```

reference: returns c1 divide by c2

comment1: returns c1 divide by c2

comment2: returns a counter that be the singular key and the give finish

comment3: returns the divide by c2

comment4: divide two counter

comment5: returns an unmodifiable counter

comment6: returns a counter with the give counter

	0	1	2	3	4
Naturalness					

Naturalness of comment1	○	○	○	○	○
Naturalness of comment2	○	○	○	○	○
Naturalness of comment3	○	○	○	○	○
Naturalness of comment4	○	○	○	○	○
Naturalness of comment5	○	○	○	○	○
Naturalness of comment6	○	○	○	○	○
Informativeness					
Informativness of comment1	○	○	○	○	○
Informativness of comment2	○	○	○	○	○
Informativness of comment3	○	○	○	○	○
Informativness of comment4	○	○	○	○	○
Informativness of comment5	○	○	○	○	○
Informativness of comment6	○	○	○	○	○