# Phase 2: Summary Construction

## [Please read carefully before attempting the task!]

## I. General Instructions:

- Attempt annotation if you are a **native speaker of English** or a **fluent speaker** who can comfortably comprehend a Stack Overflow post written in informal English.
- Annotators are required to take a 5-minutes break at each 30 minutes labeling.

## II. The Goal

This task aims to create a set of extractive summaries of Stack Overflow answers with respect to particular queries.

## III. Data Distribution

The data is in the form of a set of annotation units (AUs), while each AU consists of

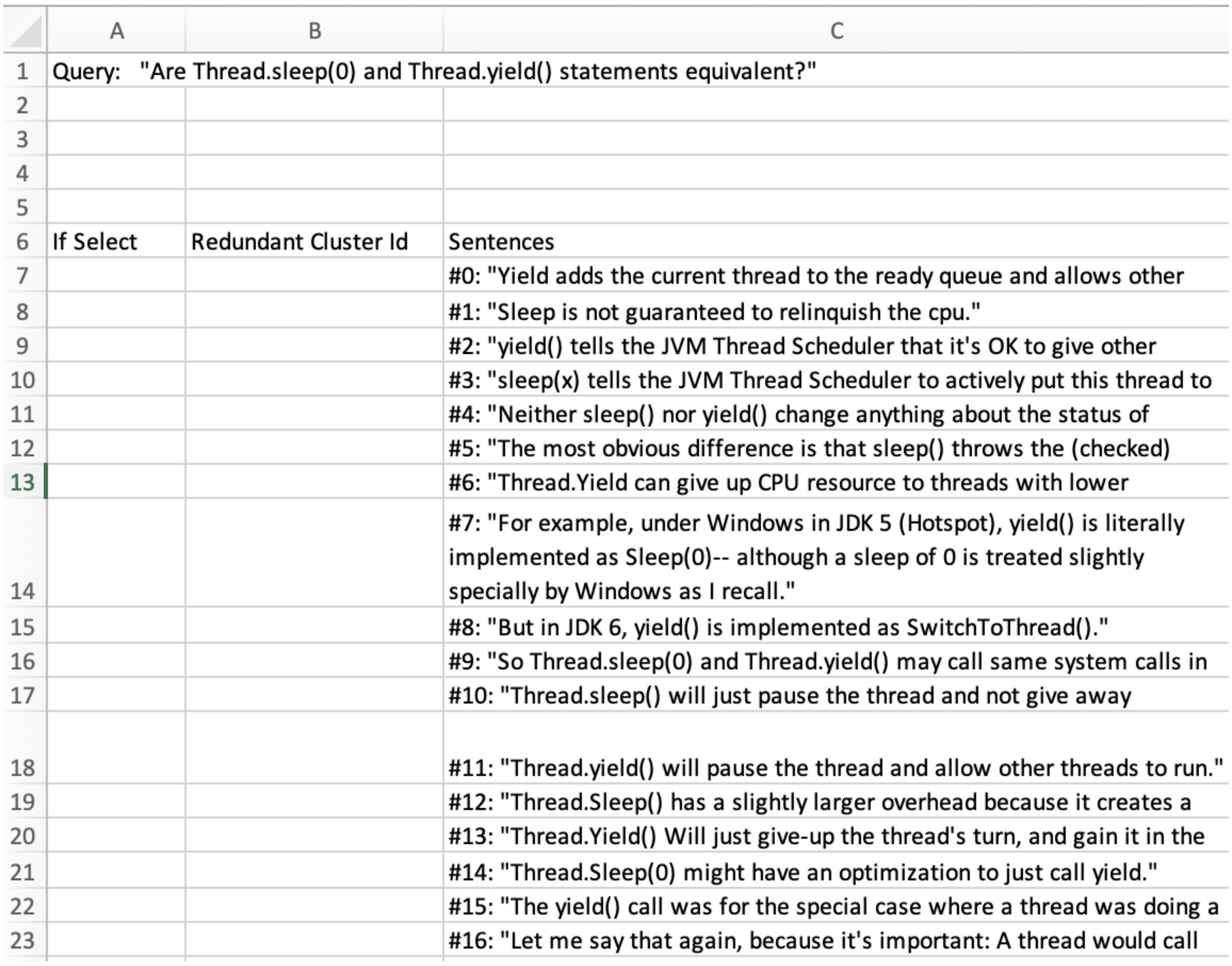
- a specific technical query
- a set of sentences deemed useful to the query

Sentences are extracted from the Stack Overflow answers with respect to the query.

## IV. Task Formulation and Labeling Format

### Input:

An annotation unit. Please see an example as below:



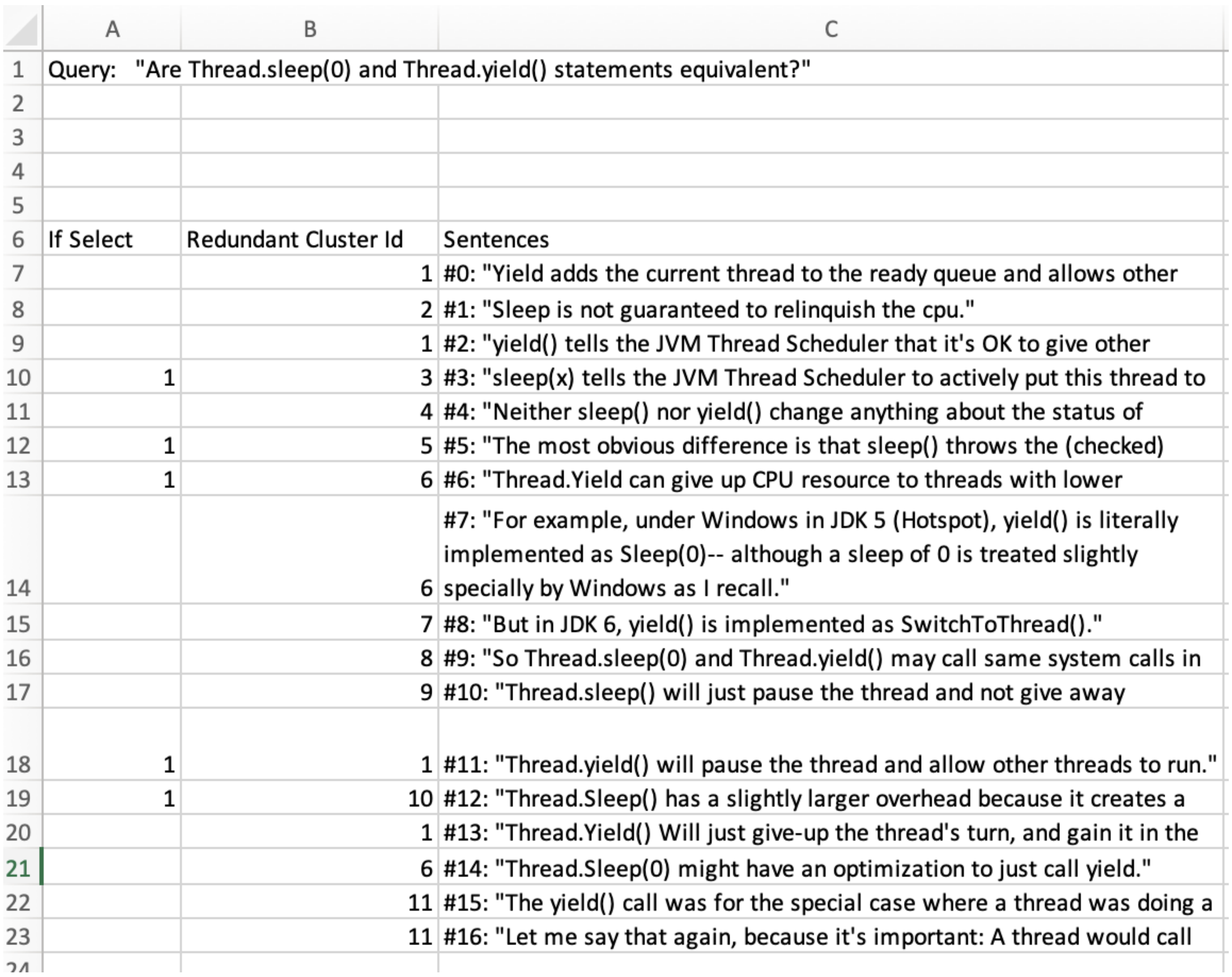
|  | A | B | C |
|---|---|---|---|
| 1 | Query: "Are Thread.sleep(0) and Thread.yield() statements equivalent?" |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 | If Select | Redundant Cluster Id | Sentences |
| 7 |  |  | #0: "Yield adds the current thread to the ready queue and allows other |
| 8 |  |  | #1: "Sleep is not guaranteed to relinquish the cpu." |
| 9 |  |  | #2: "yield() tells the JVM Thread Scheduler that it's OK to give other |
| 10 |  |  | #3: "sleep(x) tells the JVM Thread Scheduler to actively put this thread to |
| 11 |  |  | #4: "Neither sleep() nor yield() change anything about the status of |
| 12 |  |  | #5: "The most obvious difference is that sleep() throws the (checked) |
| 13 |  |  | #6: "Thread.Yield can give up CPU resource to threads with lower |
| 14 |  |  | #7: "For example, under Windows in JDK 5 (Hotspot), yield() is literally implemented as Sleep(0)-- although a sleep of 0 is treated slightly specially by Windows as I recall." |
| 15 |  |  | #8: "But in JDK 6, yield() is implemented as SwitchToThread()." |
| 16 |  |  | #9: "So Thread.sleep(0) and Thread.yield() may call same system calls in |
| 17 |  |  | #10: "Thread.sleep() will just pause the thread and not give away |
| 18 |  |  | #11: "Thread.yield() will pause the thread and allow other threads to run." |
| 19 |  |  | #12: "Thread.Sleep() has a slightly larger overhead because it creates a |
| 20 |  |  | #13: "Thread.Yield() Will just give-up the thread's turn, and gain it in the |
| 21 |  |  | #14: "Thread.Sleep(0) might have an optimization to just call yield." |
| 22 |  |  | #15: "The yield() call was for the special case where a thread was doing a |
| 23 |  |  | #16: "Let me say that again, because it's important: A thread would call |

## Output:

You are asked to create a summary **(with the top 5 most most representative sentences)** for each annotation unit to answer the corresponding query. **You also need to provide the total time spent working on this task**.

- You are asked to label the top 5 most representative sentences as **1** in the first column (i.e., If Select).
- You are asked to label the **redundant** sentences with the same number in the second column (i.e., Redundant Cluster Id).

The following is an example when you finish the labelling process for one annotation unit:



| | A | B | C |
|---|---|---|---|
| 1 | Query: "Are Thread.sleep(0) and Thread.yield() statements equivalent?" | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | If Select | Redundant Cluster Id | Sentences |
| 7 | | 1 | #0: "Yield adds the current thread to the ready queue and allows other |
| 8 | | 2 | #1: "Sleep is not guaranteed to relinquish the cpu." |
| 9 | | 1 | #2: "yield() tells the JVM Thread Scheduler that it's OK to give other |
| 10 | 1 | 3 | #3: "sleep(x) tells the JVM Thread Scheduler to actively put this thread to |
| 11 | | 4 | #4: "Neither sleep() nor yield() change anything about the status of |
| 12 | 1 | 5 | #5: "The most obvious difference is that sleep() throws the (checked) |
| 13 | 1 | 6 | #6: "Thread.Yield can give up CPU resource to threads with lower |
| 14 | | 6 | #7: "For example, under Windows in JDK 5 (Hotspot), yield() is literally implemented as Sleep(0)-- although a sleep of 0 is treated slightly specially by Windows as I recall." |
| 15 | | 7 | #8: "But in JDK 6, yield() is implemented as SwitchToThread()." |
| 16 | | 8 | #9: "So Thread.sleep(0) and Thread.yield() may call same system calls in |
| 17 | | 9 | #10: "Thread.sleep() will just pause the thread and not give away |
| 18 | 1 | 1 | #11: "Thread.yield() will pause the thread and allow other threads to run." |
| 19 | 1 | 10 | #12: "Thread.Sleep() has a slightly larger overhead because it creates a |
| 20 | | 1 | #13: "Thread.Yield() Will just give-up the thread's turn, and gain it in the |
| 21 | | 6 | #14: "Thread.Sleep(0) might have an optimization to just call yield." |
| 22 | | 11 | #15: "The yield() call was for the special case where a thread was doing a |
| 23 | | 11 | #16: "Let me say that again, because it's important: A thread would call |

# V. Hints

## V.I General pipeline

Ideally, the summary is expected to cover the most representative sentences to the given query. To achieve this, you're highly recommended to iteratively select sentences to form the summary by considering three key factors, **redundancy, clarity and importance** across all the candidate sentences by following the pipeline below:

1. Start with the first sentence and label it with the **Redundant Cluster (RC) Id** as 1.
2. Iteratively read one following sentence, and regard it as the *current sentence*.
    1. Determine whether the current sentence is redundant to any of the existing redundant cluster(s). If yes, go Step 2.2,

        otherwise, go Step 2.3.
    2. Assign the same RCId to the current sentence.
    3. Assign a new RCId to the current sentence (i.e., latest MAXIMUM Cluster Id + 1).
3. Once all the sentences are assigned with RCIds, you need to select a sentence with the highest *clarity* from each RC. To achieve this, please follow the guideline in Section V.II.I.
4. The input of this step is expected as a set of non-redundant sentences (i.e., the highest *clarity* sentences extracting from each RC). To build the summary with five sentences budget,
    1. If the size of the RCs is greater than 5, then you are advised to pick 5 of them by considering the **importance to answering the query**. To achieve this, please follow the guideline in Section V.II.II.

## V.II Concrete Examples

### V.II.I Clarity

```
Query:   "Should we always override equals?"

[ ] #1: "But yeah, if your objects lacks such distinct
attribute, then you should go ahead and check almost all
values to avoid make your programme consider two diffrent
people equal."
[ ] #2: "Because in equals implementation, you can compare the
objects against their attributes values."
```

- Both sentences are related to using *equals* function to compare the attributes value of objects. However, the first sentence #1 requires context to understand its content, e.g., "such distinct attribute". The annotators are suggested to consider the 2nd sentence with better clarity.
- Besides, if one sentence is tangled, i.e., not only carries the redundant information but also other information, this sentence

shouldn't be regarded as redundant.

- If it's difficult to judge the clarity for pair of sentences, please consider labeling the more concise (e.g., less words) sentence as better clarity one.

### V.II.II Importance of Information

```
Query:   "Should we always override equals?"
[1] #1: "To test whether two objects are equal in the sense of equivalency (containing the same information), you must override the equals() method.You should always override the equals() method if the identity operator is not appropriate for your class."   --> More important
[] #2: "Because in equals implementation, you can compare the objects against their attributes values."   --> Less important
```

- Although both sentences are clear and useful to the query, you are suggested to label the 2nd sentence #2 as less important. Because in the 2nd sentence, the relationship between "equals implementation can compare the attributes value" and "whether we should override equals" remains uncertain (or weak).