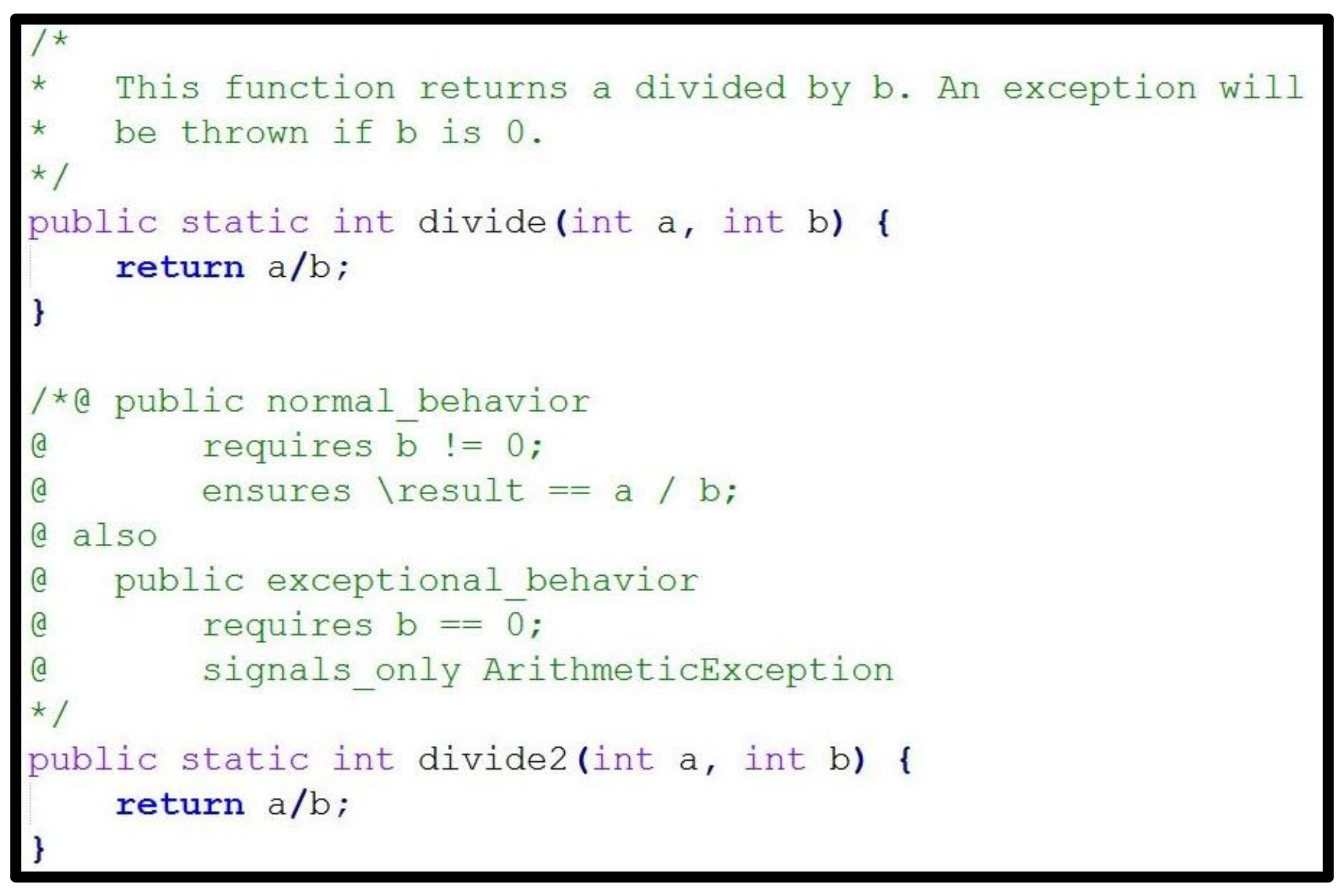
Socio-Technical Solution to Large-Scale Formal Specification Mining

May1620
Alex Dana, Cody Hanika
Client/Adviser: Professor Hridesh Rajan

Problem:

- ❏ Formal specifications are not widely available to programmers
- ❏ Many programmers do not know how or why to use formal specifications
- ❏ It can be difficult to write new formal specifications without a lot of experience using them (even then it is quite time consuming)
- ❏ It is difficult to automatically generate formal specifications for the code one is writing



```
/*
*   This function returns a divided by b. An exception will
*   be thrown if b is 0.
*/
public static int divide(int a, int b) {
    return a/b;
}

/*@ public normal_behavior
@       requires b != 0;
@       ensures \result == a / b;
@ also
@   public exceptional_behavior
@       requires b == 0;
@       signals_only ArithmeticException
*/
public static int divide2(int a, int b) {
    return a/b;
}
```

Informal vs. Formal JML Specifications

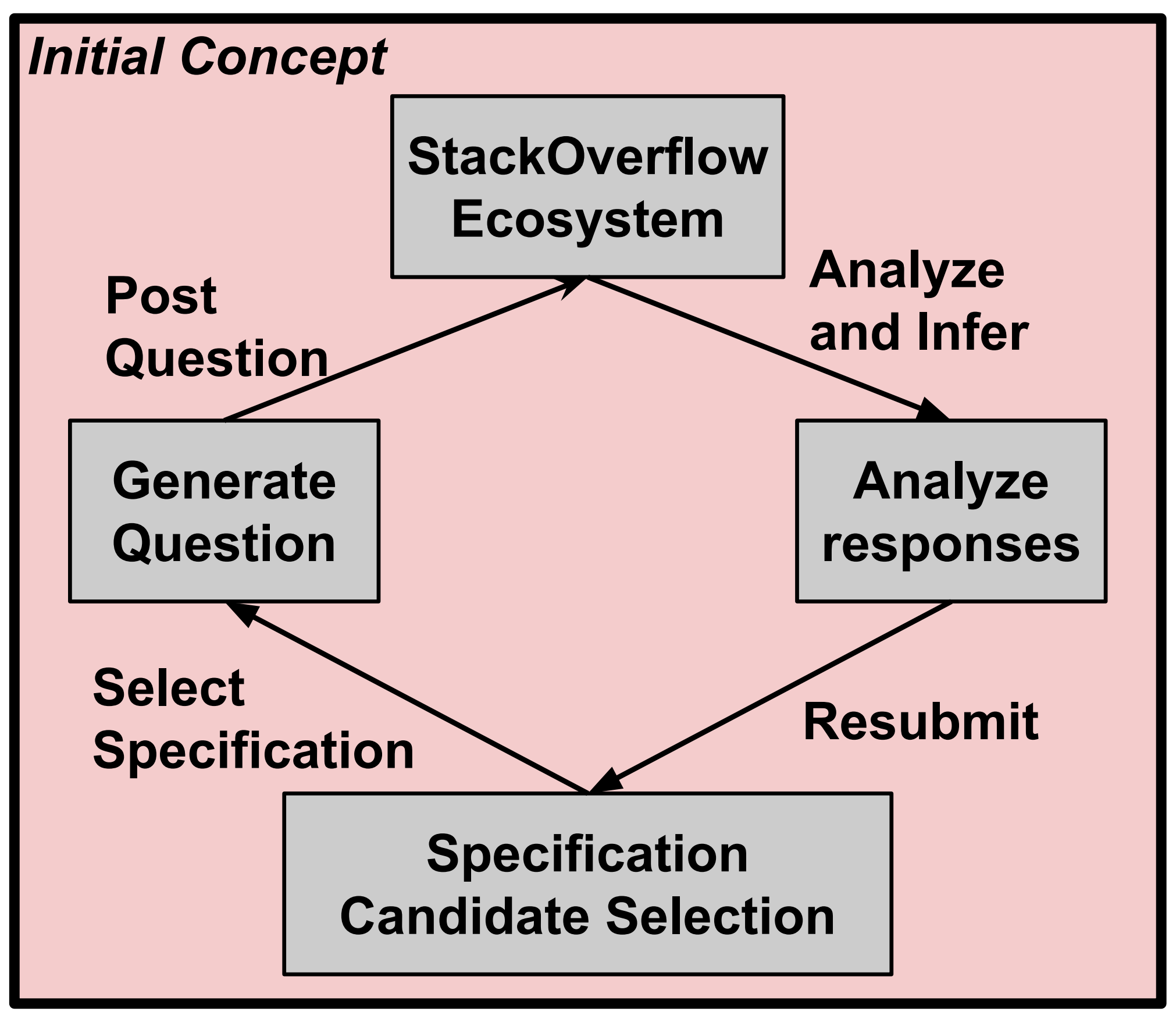
Operating Environment:

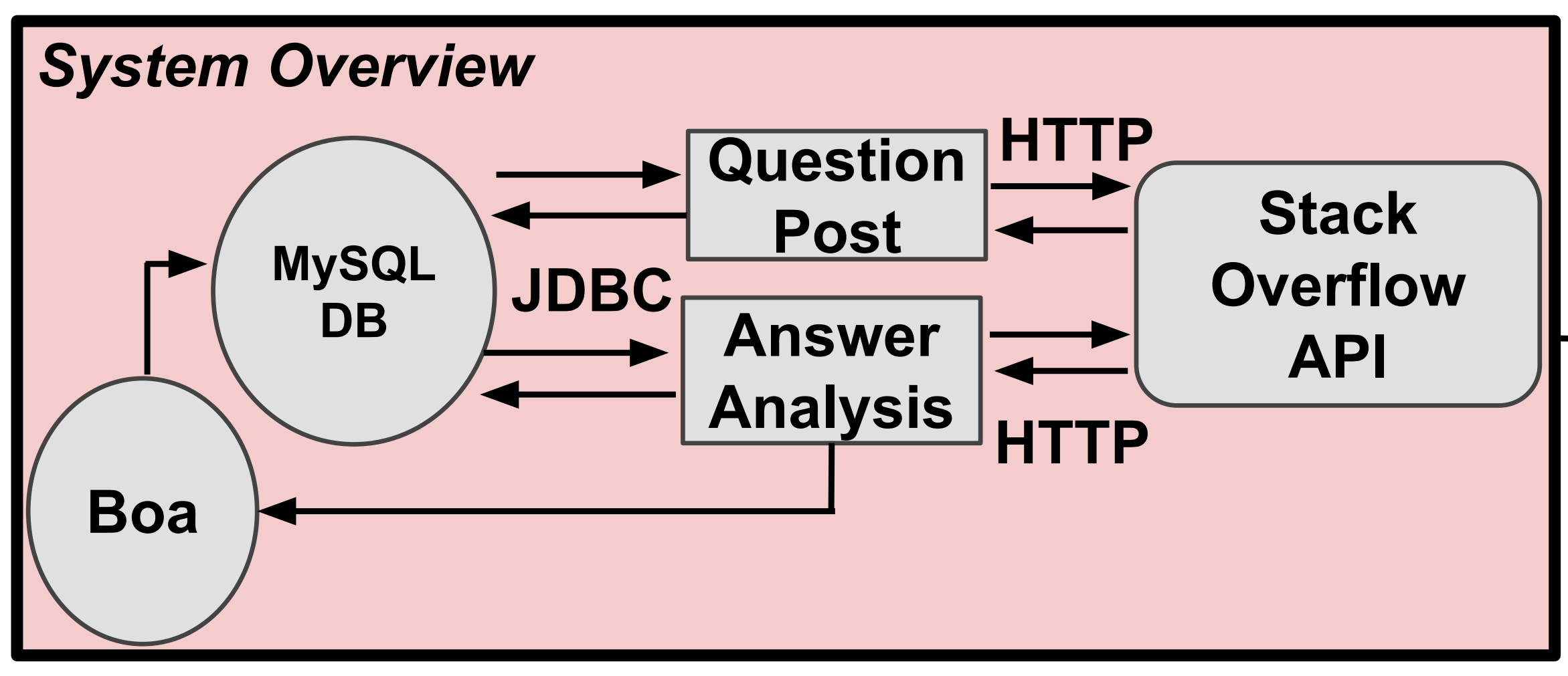
Our main Java systems will be platform independent with our MySQL database running on Linux.

Approach:

- ❏ We tackled this problem using 2 unique Research Stages
  - ❏ **Stage 1** - Attempt to mine JML (Java Specification Language) specifications from StackOverflow questions
    - ❏ We generate these questions via http requests using the SO API
  - ❏ **Stage 2** - Use current javadoc to generate JML utilizing a natural language processor
    - ❏ The Stanford Natural Language Processor (SNLP) was chosen

Stage 1





Question Post: Retrieve unfinished spec from database (JDBC). Java http request to generate and post question via SO API.
Answer Analysis: Retrieve in progress spec. Java http request to retrieve question via SO API. Analyze using Java.
Database: MySQL database to store specs in need of improvement as well as specs currently posted.
Boa System: Large software repository knowledge base.
StackOverflow API: StackOverflow provides a robust API for posting, answering, and retrieving questions.

Functional Requirements:

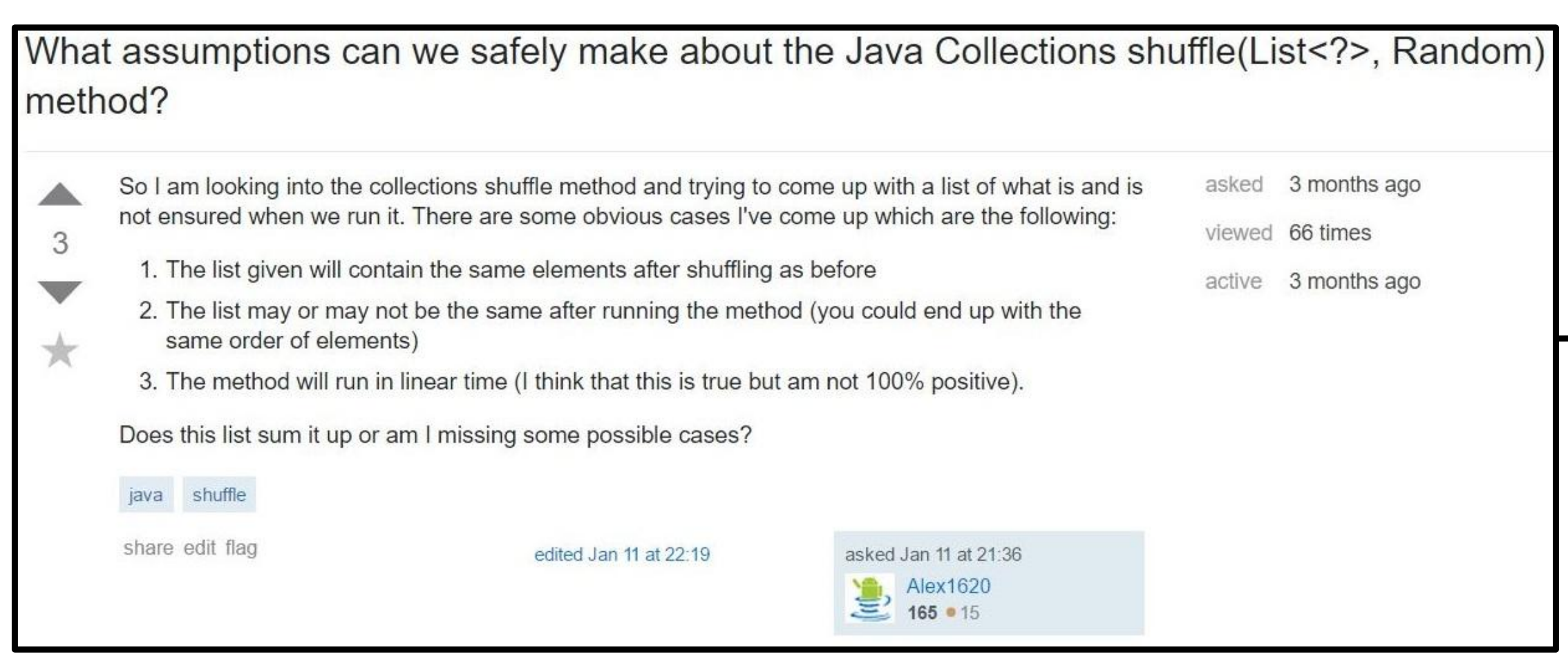
- ❏ Generate valid JML specifications
- ❏ Automatically generate SO questions

Non Functional Requirements:

- ❏ Run continuously
- ❏ Little/No user input

Test Plan:

JUnit used to ensure the components of our system interfaced properly. Manual tests to validate http request success.



What assumptions can we safely make about the Java Collections shuffle(List<?>, Random) method?

3

So I am looking into the collections shuffle method and trying to come up with a list of what is and is not ensured when we run it. There are some obvious cases I've come up which are the following:

1. The list given will contain the same elements after shuffling as before
2. The list may or may not be the same after running the method (you could end up with the same order of elements)
3. The method will run in linear time (I think that this is true but am not 100% positive).

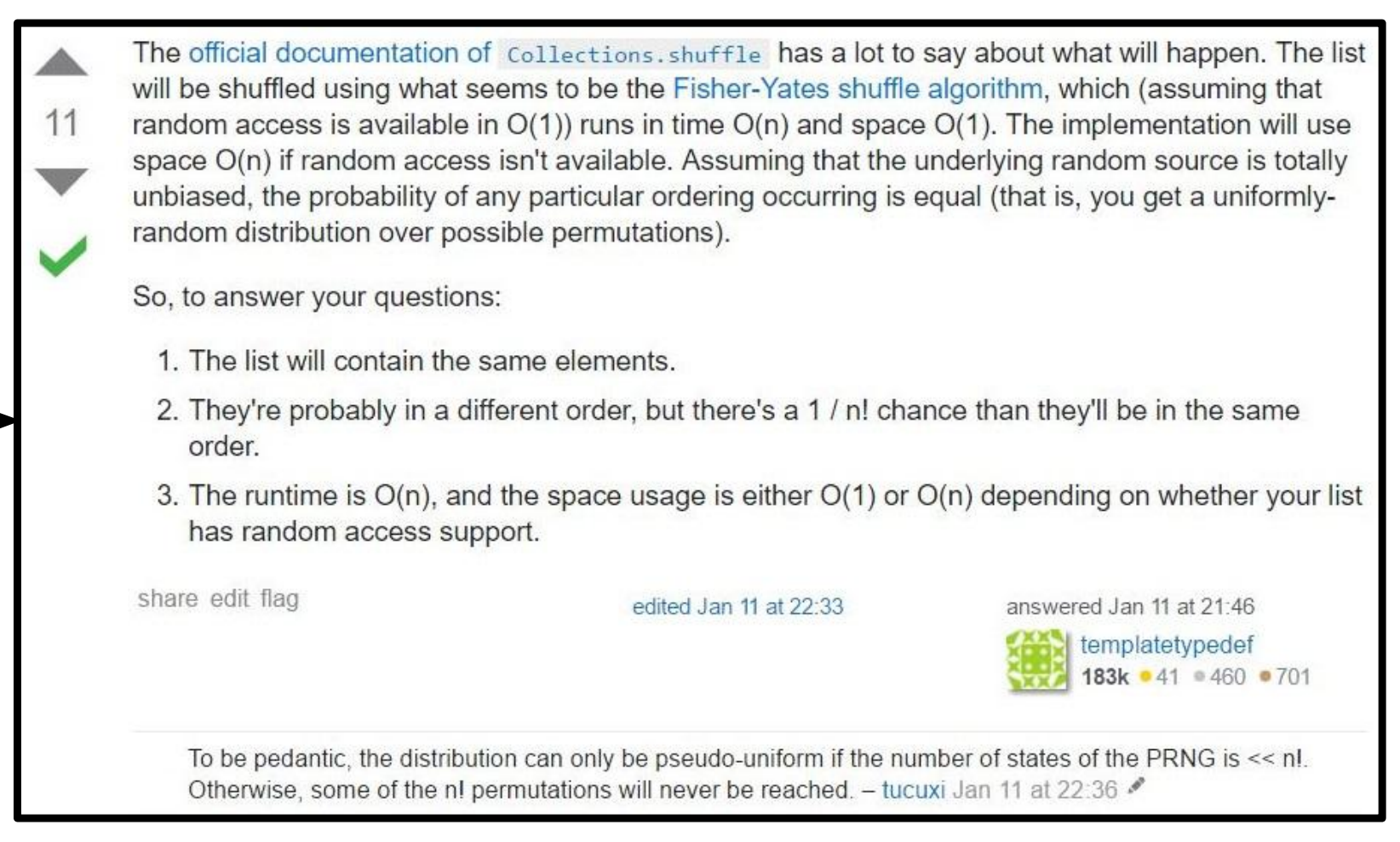
Does this list sum it up or am I missing some possible cases?

java shuffle

share edit flag — edited Jan 11 at 22:19 — asked Jan 11 at 21:36 Alex1620 165 ●15

asked 3 months ago
viewed 66 times
active 3 months ago

Question asked regarding the 'shuffle' method



11

The official documentation of `Collections.shuffle` has a lot to say about what will happen. The list will be shuffled using what seems to be the Fisher-Yates shuffle algorithm, which (assuming that random access is available in O(1)) runs in time O(n) and space O(1). The implementation will use space O(n) if random access isn't available. Assuming that the underlying random source is totally unbiased, the probability of any particular ordering occurring is equal (that is, you get a uniformly-random distribution over possible permutations).

So, to answer your questions:

1. The list will contain the same elements.
2. They're probably in a different order, but there's a 1 / n! chance than they'll be in the same order.
3. The runtime is O(n), and the space usage is either O(1) or O(n) depending on whether your list has random access support.

share edit flag — edited Jan 11 at 22:33 — answered Jan 11 at 21:46 templatetypedef 183k ●41 ●460 ●701

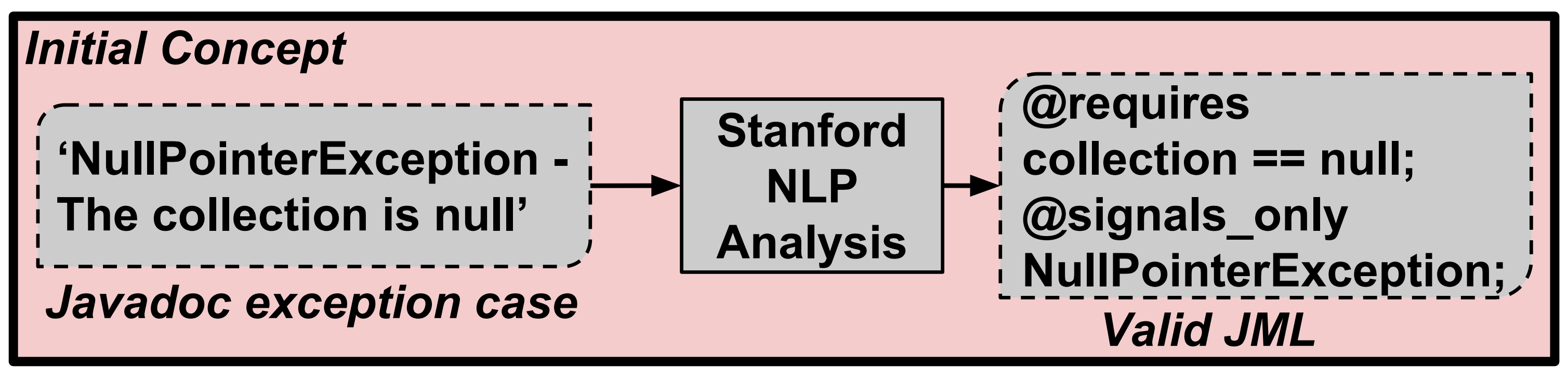
To be pedantic, the distribution can only be pseudo-uniform if the number of states of the PRNG is << n!. Otherwise, some of the n! permutations will never be reached. – tucuxi Jan 11 at 22:36

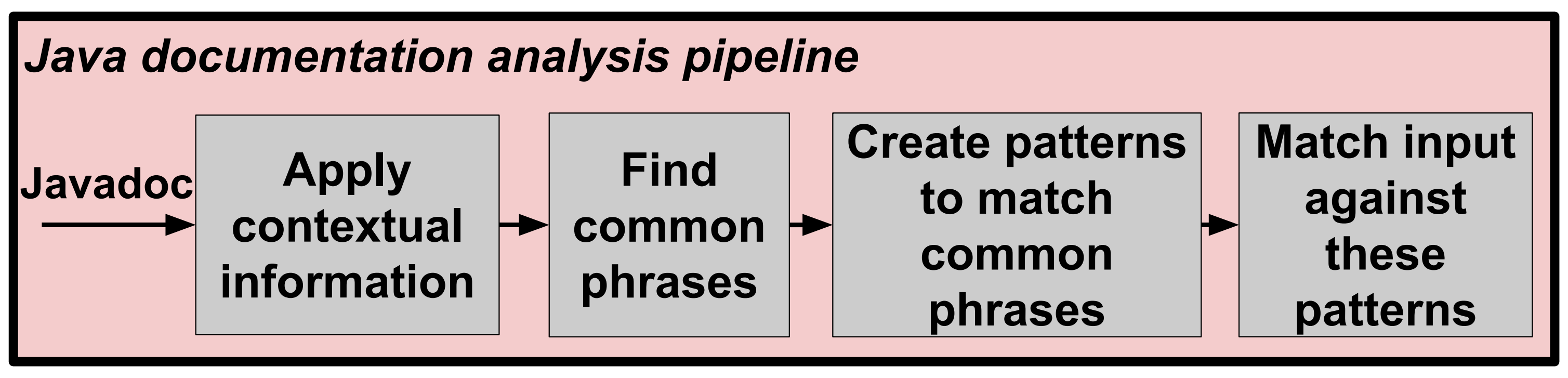
Valid but non-structured answer

Results:

Research concluded that stage 1 was not a viable solution to the problem. A general lack of quality answers on StackOverflow led us to abandon this approach in favor of stage 2.

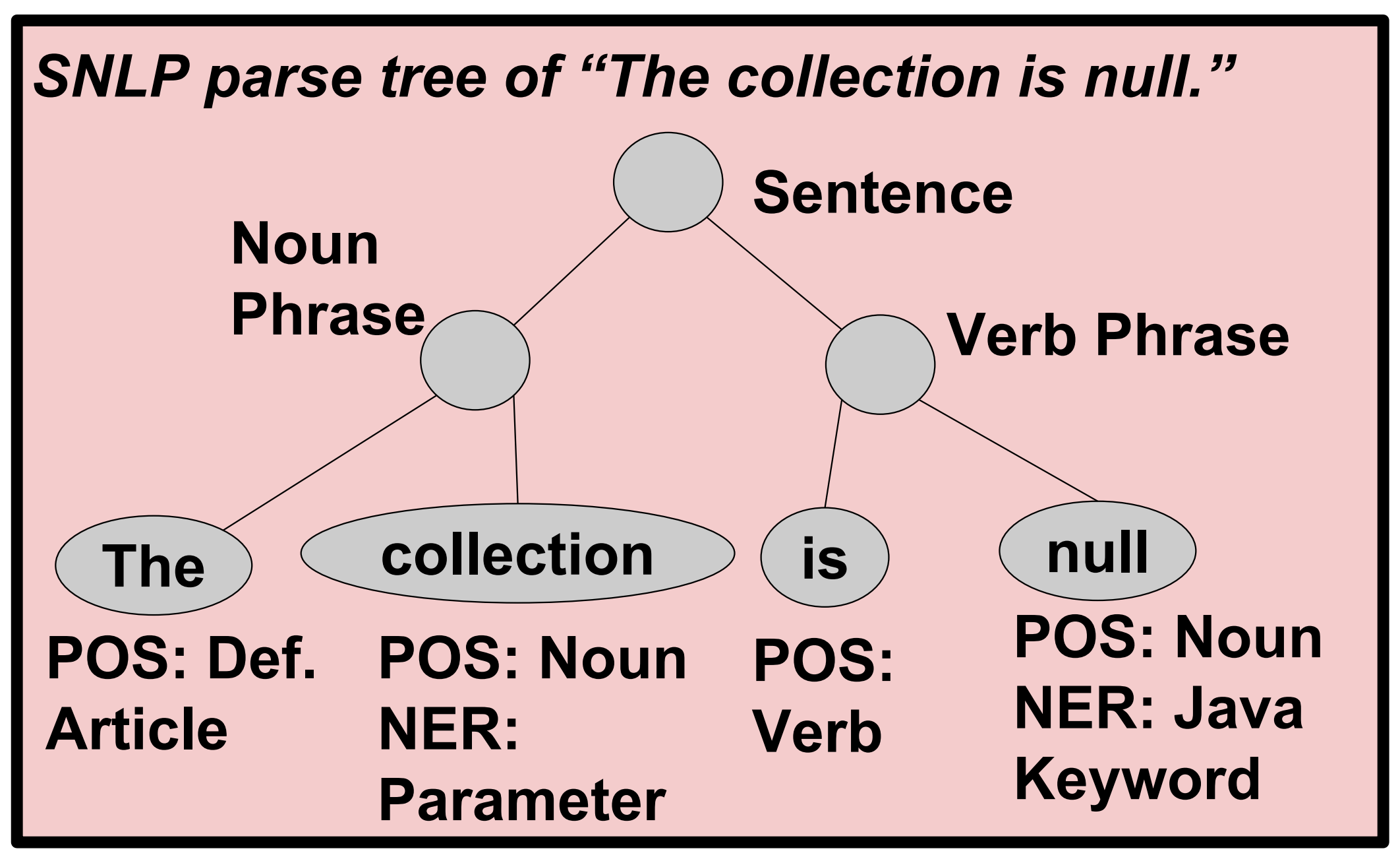
Stage 2


Initial Concept
'NullPointerException - The collection is null'
Javadoc exception case
Stanford NLP Analysis
@requires collection == null;
@signals_only NullPointerException;
Valid JML


Java documentation analysis pipeline
Javadoc
Apply contextual information
Find common phrases
Create patterns to match common phrases
Match input against these patterns

Our Process:

- ❏ Parse javadoc of jdk
- ❏ Create a parse tree via the analysis pipeline
- ❏ Extract frequent parse trees to discover common informal specifications
- ❏ Create 'tokensregex' rule files to generate JML from these informal specifications



Functional Requirements:

- ❏ Generate valid JML specifications
- ❏ Obtain javadoc by parsing source code

Non Functional Requirements:

- ❏ Analysis runs in a 'reasonable' amount of time (less than 2 minutes)

Parse Tree:
We create a parse tree of the javadoc using SNLP. All subtrees are hashed and then we look for common tree structures.

Tokensregex Rule Files:
We create 'rules' using regular expressions to tell SNLP how to handle certain groups of words, POS, NERs, etc. which we use to generate JML.

Test Plan:

Manually ran analysis over Java's standard library to identify percentage of methods we found matches for. Then verify how many of them were valid.

Results:

Using SNLP we were able to successfully generate JML for certain types of Exceptions, such as NullPointerException. We had less success with documentation that didn't describe a boolean expression.