Report for assignment 3

Updates done after feedback from TA
Section Coverage report: It includes links to two drive folders containing screenshots of the branch cover of our functions before and after improving. As Well as the whole cobertura report with instructions on how to download it and see it.
Included a link to an textfile containing the git diff of our branch with all test cases.
Put our name on the function that we fixed and refactored.
Aiming for P+: Julia, Carl, Elisabeth, Victor

Project

Json interator (jsoniter)
an improved JSON parser for Java and Go

URL: https://github.com/json-iterator/java

Onboarding experience

How good is the "onboarding" documentation?
1. How easily can you build the project? Briefly describe if everything worked as documented or not:
(a) Did you have to install a lot of additional tools to build the software?
No, all that is required to build the project is Maven.

(b) Were those tools well documented?
No, the fact that the project was using Maven was not documented at all.

(c) Were other components installed automatically by the build script?
Yes, Maven installed all the required dependencies upon building.

(d) Did the build conclude automatically without errors?
The code compiled without errors, but there are several test failures and a multitude of errors.

(e) How well do examples and tests run on your system(s)?

As mentioned, there are test failures, although a majority of tests passes. There is a folder called “demo” which is its own Maven project, which would serve as an example on how to use the project code, but it won’t build as it can’t find the `jsoniter` dependency.

2. Do you plan to continue or choose another project?
We plan to continue with this project.

Complexity

1. What are your results for ten complex functions?

(get the result via Lizard command “lizard -s cyclomatic_complexity {path}\java”)
==========
NLOC CCN token PARAM length location
---
1. 105 28 799 2 110 IterImpl::readStringSlowPath@217-326@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\IterImpl.java
2. 99 27 725 2 104 IterImplForStreaming::readStringSlowPath@391-494@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\IterImplForStreaming.java
3. 110 26 764 1 128 CodegenImplObjectStrict::genObjectUsingStrict@24-151@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\CodegenImplObjectStrict.java
4. 111 24 692 2 111 GsonCompatibilityMode::createDecoder@335-445@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\extra\GsonCompatibilityMode.java
5. 73 23 527 2 77 CodegenImplNative::genReadOp@195-271@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\CodegenImplNative.java
6. 57 21 668 2 57 Parsed::parse@138-194@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\spi\OmitValue.java
7. 49 20 273 1 50 IterImplForStreaming::readNumber@571-620@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\IterImplForStreaming.java
8. 58 18 394 1 63 Config::updateBindings@394-456@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\spi\Config.java

9. 60 18 391 1 62 Codegen::chooseImpl@120-181@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\Codegen.java
10. 36 18 148 1 36 IterImplSkip::skip@19-54@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\IterImplSkip.java

Extra. 34 16 403 0 54 Base64::decodeFast@203-256@C:\Users\elisa\OneDrive\Dokument\SchoolProgramming\soffunCode\Open_source\java\src\main\java\com\jsoniter\extra\Base64.java

Complexity coverage

1. Did all methods (tools vs. manual count) get the same result?
    - **IterImpl::readStringSlowPath**
        - Tools: 28
        - Elisabeth: 28
        - Victor: 28
    - **GsonCompatibilityMode::createDecoder**
        - Tools: 24
        - Carl: 24
        - Hemen: 22
    - **Config::updateBindings**
        - Tools: 18
        - Elisabeth: 17
        - Victor: 17
    - **CodegenImplNative::genReadOp**
        - Tools: 23
        - Hemen: 22
        - Julia: 23
        - Elisabeth: 23
    - **Parsed::parse**
        - Tools: 21
        - Hemen: 19
        - Victor: 20
        - Elisabeth: 20
    - **IterImplForStreaming::readNumber**
        - Tool: 20
        - Julia: 20 (includes default in switch statement)
        - Carl: 20 (includes default in switch statement)

2. Are the results clear?

To measure cyclomatic complexity, we counted if-statement (including ternary operator), else if statement (did not count else statement), cases in switch-statement, for and while loop and try-catch statement (one try-catch counts as +1). We did not add return, since it does not create another branch. By regarding each case in a switch-case statements as its own branch even when fallthrough (no break statement) was used we got a similar result to Lizard. In general our results are really close to the results from lizard.

3. Are the functions just complex, or also long?

Overall, most of the function are just complex and not so long. Most of the lines of codes for these functions are if/else statements that increase the complexity number, but not necessarily the number of lines of code.

4. What is the purpose of the functions?

1. **IterImpl::readStringSlowPath**

No documentation. I assume that this function is a parser that test a few basic rules on the JSON request.

2. **IterImplForStreaming::readStringSlowPath**

There was no document. From code, guessing the function is a parser that tests a few basic rules on the JSON request.

3. **CodegenImplObjectStrict::genObjectUsingStrict**

Very difficult to say as there is no general documentation for the function, and I do not have much knowledge of json.

4. **GsonCompatibilityMode::createDecoder**

The function returns a Decoder object which is able to decode a specific type, e.g. int or String. For each type the function also overrides a Decoder function.

5. **CodegenImplNative::genReadOp**

Given a certain type and cachedKey it returns where the reading operation should be done.

6. **Parsed::parse**

a function is used in an interface called "OmitValue", it parses the given default value according to value type and returns object with new default value and speciell code. Not documented

7. **IterImplForStreaming::readNumber**

Takes a number as a string and parses it into a custom object called numberChars that contains the number as an char array, if it contains a dot and its length.

8. **Config::updateBindings**

It's a private function used in updateClassDescriptor. There was no document. From code, guessing the function update the variables with type Binding based on the values from ClassDescriptor.

9. **Codegen::chooseImpl**

No documentation. From code, guessing the function choose the appropriate type for Impl.

10. **IterImplSkip::skip**

There was no document. From code, guessing the function is a parser that skips reading some symbols based on the input.

Extra. Base64::decodeFast

There was no document but it had some short comments to explain part of the code. From these comments, the code and the function name, guessing the function is a decode base64 format text to ASCII text.

5. Are exceptions taken into account in the given measurements?

Lizard seem to take into account try/catch blocks, but not one-line `throw new exception` statements.

6. Is the documentation clear with regard to all the possible outcomes?

There is no documentation on any function, not in the code as comments nor on the documentation website.

Refactoring

1. Plan for refactoring complex code:

(Carl) GsonCompatibilityMode::createDecoder

Create a new function for each data type (7 in total), i.e. split the function into 8 new ones.

(Victor) IterImpl::readStringSlowPath

This function is a huge for loop. It is a bit difficult to divide it into smaller functions. But, in order to do that, we can put the switch case in another function. We first need to save the context using the new class switchCaseContext, then we execute the switch case code in another function, and finally we update the variables in the main function with the context values.

(Hemen) Parsed::parse

The function is already using guard and early return when nesting if-else statements so no better plan could be used in this function according to the complexity of its definition, maybe using more default value combined with || operator or divide the function into two functions.

(Elisabeth) Config::updateBindings

This function consists of many if-else statements in a large for-loop. This can be separated by creating many private functions for if-else statements in the loop (this does not include every if-else statement, since then it will be too many functions). By analysing the code, it would be appropriate to divide the if-else statements to 6 functions (7 including the updateBindings function).

(Julia) IterImplForStreaming::readNumber

This function has a complexity number of 20. This is mostly because it parses a number and has a case for each valid symbol in a number. One could argue that it has essential complexity because a number could contain many symbols and each symbol needs to be handled. However a lot of symbols in the number are handled but have their own case statement. Thus a refactoring plan is to group symbols that are handled the same into one branch.

2. Estimated impact of refactoring (lower CC, but other drawbacks?).

(Carl) GsonCompatibilityMode::createDecoder

CC would go down by about two thirds when splitting the function into 8 parts. In total there will be more code lines, since every new function need a signature etc. Although the function itself might be easier to read, overall the whole java file might become messier due to the increased number of functions.

(Victor) IterImpl::readStringSlowPath

This can decrease the complexity number from 28 to 13 in the main function but implies to store the context, which can increase execution time.

(Hemen) Parsed::parse

The CC unfortunately is little bit difficult to reduce more than 19

(Elisabeth) Config::updateBindings

The CC will definitely go down, since we decrease the number of if-else statements in each function. The function itself will also be easier to read. However, the total code line will increase and the java file will become messier due to the increased number of functions. The cyclomatic complexity for each functions is following (beside the first function, the rest is the separated private functions):

- Config::updateBindings: 7
- Config::jsonIgnoreIf: 4
- Config::jsonUnwrapperIf: 2
- Config::jsonPropertyIf: 2
- Config::annotationIf: 3
- Config::setSetterNames: 4
- Config::setGetterNames: 4

(Julia) IterImplForStreaming::readNumber
When parsing a number there is a switch-case statement that contains 15 cases + 1 one default case. So in total 16 branches. However there are only three different ways a symbol is handled. So by grouping similar symbols in an if-statement, 16 branches could be turned into 3. This would reduce the complexity by 16-3=13 branches. Thus reducing the count of 20 branches to 7.

3. Carried out refactoring (optional, P+):

(Carl) createDecoder: refactored createDecoder (with branch coverage and more tests)
(Julia) readNumber link:refactored readNumber (with branch coverage and more tests)
(Victor) readStringSlowPath link:refactored readStringSlowPath
(Elisabeth) updateBindings: link: refactored updateBindings (with branch coverage and more tests)

Coverage

Tools

For this project we used Cobertura, which was used in the project we chose. There were some different problems with installing and integrating Cobertura in our build environment, e.g. it could not find tools.jar (a file Cobertura uses) in Java if the version is higher than 8 since Java has merged this file with other files in Java 9 or higher. The usage of Cobertura was pretty easy, since it was just to run Test on maven to get the results. The results were also organized in a clear way. However, the documentation linked on their GitHub site was not enough to understand how to use Cobertura and it did not give any documentation of possible errors. We also used OpenClover and JaCoCo because it integrated well with some of our editors.

Your own coverage tool

First a counter is added to every branch that has a unique branch id. This counter will increment every time it goes though the corresponding branch. These counters are then saved at the field variable “branchCounters” as an Integer array for respective function. These counters are then logged by using FileWriter and saved in the folder logFiles with the file name “{function name}Log.txt” (e.g., function name “foo”, file name “fooLog.txt”).
Below is the link to a git repository for our own manual coverage measurement tool. Also added for specific links directly to the main code for the tool, the functions that are tested and the result in folder LogFiles.

Git repository for the tool: https://github.com/DD2480-Group26/java/tree/issue/18

Link to the main code for the tool

Link to functions used in the tool

Link to respective function:
- readStringSlowPath
- readNumber
- updateBindings
- parse
- createDecoder

Link to the manual branch coverage report:
https://github.com/DD2480-Group26/java/tree/issue/18/logFiles

What kinds of constructs does your tool support, and how accurate is its output?
The tool can support any type of structure, since it is used manually. Thus you can choose whether you want to count e.g. ternary operators as a branch or not. The output shows how many times each manually added branch has been executed.

Evaluation

1. How detailed is your coverage measurement?
The coverage tool only shows if one branch has been tested or not. We could have checked how many times this branch has been tested.

2. What are the limitations of your own tool?
This tool is not really scalable. We need to add some code in each branch of every function that we want to test. Then for big functions the output can be difficult to read.

3. Are the results of your tool consistent with existing coverage tools?
Yes, for most cases, our tool gives the same result as other coverage tool.

Coverage improvement

(Carl) createDecoder (Link):
- Tested:
    A. Whole Date.class branch (except throw statement).
    B. Boolean as String.class with a true boolean.
    C. Boolean.class with a null object.
    D. Long.class with a null object.
    E. Int.class with a null object.
    F. Float.class with a null object.
    G. Double.class with a null object.
- Untested:

1. Boolean as String.class with a false boolean (ternary operator branch).
2. String.class with a null object.
3. Boolean.class with a boolean object.
4. Long.class with a long object.
5. Int.class with an int object.
6. Float.class with a float object.
7. Double.class with a double object.
8. Every throw statement at the end of each *.class branch.

- Test cases added for:
  - 1, 2, 3, 4, 5, 6 in file TestGson.java (added tests are commented).

(Elisabeth) updateBindings (Link):

- Tested:
  1. Values related to all kind of Binding variables (e.g. jsonnore, jsonUnwrapper, jsonProperty, binding.annotations, field, etc.)
  2. Values specific related to the Binding variables setter and getter (
- Untested:
  3. If binding.field.getName() is not equal to getter.name (both variables for binding type)
  4. If getters (an arrayList) is empty
- Test cases added
  - Tested point 3 in the test case TestUpdateBindingsBranch21 in File TestConfig
  - Tested point 4 in the test case TestUpdateBindingsBranch22 in File TestConfig

(Elisabeth) decodeFast (Link):

- Tested:
  1. Decode normal ASCII text
  2. base64 formatted string with none “=” at the end
  3. Decode base64 string that length in bytes can be divided by 3. (there is a for-loop in the code that runs 3 bytes at time. In the comments it’s written that it runs for all bytes beside the 0-2 last bytes)
- Untested:
  4. Illegal chars at the start of the string
  5. Illegal chars at end of the string
  6. Empty base64 formatted string
  7. The last 1-3 bytes if they are left after the for-loop mentioned in point 2. (it’s possible with 3 bytes since it also includes “=” that they have trimmed away in the for-loop mentioned in point 2)
  8. base64 formatted string with two “=” at the end
  9. base64 formatted string with one “=” at the end
  10. base64 formatted string that is longer than 76 bytes
  11. If the string includes line separators
- Test cases added:
  - Added test cases in the file TestBase24 for point 4-10 with the following names:
    - Point 4: test_illegal_chars_from_start()

- Point 5: test_illegal_chars_from_end()
- Point 6: test_empty()
- Point 7: test_decode_last_1_to_3_bytes()
- Point 8: test_two_equals()
- Point 9: test_one_equals()
- Point 10: test_more_than_76_bytes()

(Julia) readNumber (Link)

- Tested:
    1. Parsing the maximum double value
- Untested:
    1. Reading numbers containing more than 32 symbols successfully.
    2. Reading empty strings as numbers should give an empty char array.
    3. Reading negative floating numbers successfully.
    4. Parsing number containing not allowed char should end parse.
- Test cases added:
    - Tested point 1 in the test case testReadMoreThan32BitNumber in File IterImplForStreamingTest
    - Tested point 2 in the test case testEmptyNumber in File IterImplForStreamingTest
    - Tested point 3 in the test case testNegativeFloat in File IterImplForStreamingTest
    - Tested point 4 in the test case testParseEndOnNotAllowedChar in File IterImplForStreamingTest

(Hemen) Parse (Link):

- Tested:
    - The interface “OmitValue” classes are tested all.
- Untested:
    - The parse function in the “Parsed” class was not tested.
- Test cases added:
    - Tested Parsed class and parse function to make sure that it is working and getting the right value.

(Victor) readStringSlowPath (Link)

- Tested:
    1. Mosts of the cases where buffer contains ““‘
    2. Case where buffer contains ‘\\’

- Untested:
    1. Case where buffer contains invalid unicode
    2. The case where iter.isExpectingLowSurrogate is true.
- Test cases added:
    - In file testIterImpl:
        - Test 4: Test case where buffer contains ‘\\b’
        - Test 5: Test case where buffer contains ‘\\n’
        - Test 6: Test case where buffer contains ‘\\t’
        - Test 7: Test case where buffer contains ‘\\f’

Branch with all added tests: https://github.com/DD2480-Group26/java/tree/issue/9

File with gif diff for branch with added test compared to master:
https://docs.google.com/document/d/1iyP1eLyuFS1hbiEI9IeWugn4XCJgQbQxasrdjXEu5kk/edit?usp=sharing

Coverage report

Report of old coverage:

- Google folder for screenshots of the coverage improved functions **BEFORE IMPROVEMENT**: Link
- Git repository: Link (How to open: pull the branch and open the file target/site/cobertura/index.html in a browser to see the report)

Report of new coverage:

- Google folder for screenshots of the coverage improved functions: Link
- - Git repository: Link

Self-assessment: Way of working

Current state of the group is probably somewhere between “In Use” and “In place” according to the Essence standard. This is assessed using the checklist for Way-of-working on page 60 “Essence -kernel and language for software engineering methods,” (Omg.org, 2018). We have established and used tools that we earlier decided upon. Everyone utilises tools like GithHub and junit testing to complete work within the context of the assignment. Thus we could say that these technologies are supported by the team. Using communication tools like GitHub and discord we are able to collaborate. Our ways of communication are established and something that we have improved during the course. Because we sometimes iterate the tools between assignments there will always be some in between time where not everyone will have access to the tools thus we are probably not in “In place” yet but in “In Use”. To further improve we need to improve the process of iterating tools used to make it easier for everyone to support new tools like when we changed to Maven from Gradle.

Overall experience

What are your main takeaways from this project? What did you learn?
Is there something special you want to mention here?

The project overall was very learning and was something new we worked on. Being able to read, understand code in existing large-scale projects was not easy but it enhanced our skills by trying to run and build the project using specific tools and development

environments as well as using several tools to analyze and improve code was really interesting.