[arc42] Documentation of asdf interpreter

1. Introduction and Goals

In this project it is all about building a Interpreter for the esolang asdf. The goal is to interpret asdf to java code, that can be easily run by the jdk. I have done this, because asdf is similar to brainfuck and as such easy to understand.

1.1. Requirements Overview

The requirements of this project are: - Build a interpreter - Build a interpreter, that can output real characters on the console

1.2. Quality Goals

Quality Goal	Explanation
Output text on console	Text can be better understood than numbers, so number ouput is only outputed when the user wants to.
Enable the user to give the program its own code	Interpret the input through the command line.

1.3. Stakeholders

The stakeholder of this project is Mr. von Kaenel.

2. Architecture Constraints

Constraint	Explanation, Background
Implementation in Java	Implementation in Java, Version 16 and it should also run on newer versions.
Don’t use any not-standard Java packages	For the implementation it is not allowed to use a additional library to convert an (ascii) number into a string.
Github	This project is versioned with github.
Topic	Build a transpiler, compiler or interpreter

3. Conventions

Constraint	Explanation, Background
Documentation	The documentation has to be done with the arc42 template and asciidoc.
Language	The language of this project is English, because the user of this program is good in this and it is easier to further develop the code.

4. System Scope and Context

4.1. Business Context

```
Dot Executable: /opt/local/bin/dot
File does not exist
Cannot find Graphviz. You should try

@startuml
testdot
@enduml

 or

java -jar plantuml.jar -testdot
```

4.2. Technical Context

4.2.1. Asdf characters

Character	Explanation
a	Sets toogle flag to true and current location in memory + 1 as well as printing the second memory position, if called twice on position 1.
s	If the toggle flag is false go one memory-position to the right. If its true then set it to false and move the memory pointer to the left.
d	Opening bracket and runs the code till the next f. Sets the toggle flag to false.
f	Closing bracket and makes, that the execution continues at the corresponding d. Sets the toggle flag to false.
g	Substracts -1 at the current location in memory.
h	Sets the toggle flag to false.

5. Solution Strategy

5.1. Construction

Asdf is a Java program. It consists of the following steps:

It preprocesses the code and puts it into the right order. After that it runs the code successively and prints the ouput to the command line.

5.2. Connection

Asdf doesn’t have a graphical interface. Instead it gets the user input from in- and output.

6. Building Block View

6.1. Important Interfaces

6.1.1. Syntax handling

Purpose/Responsibility

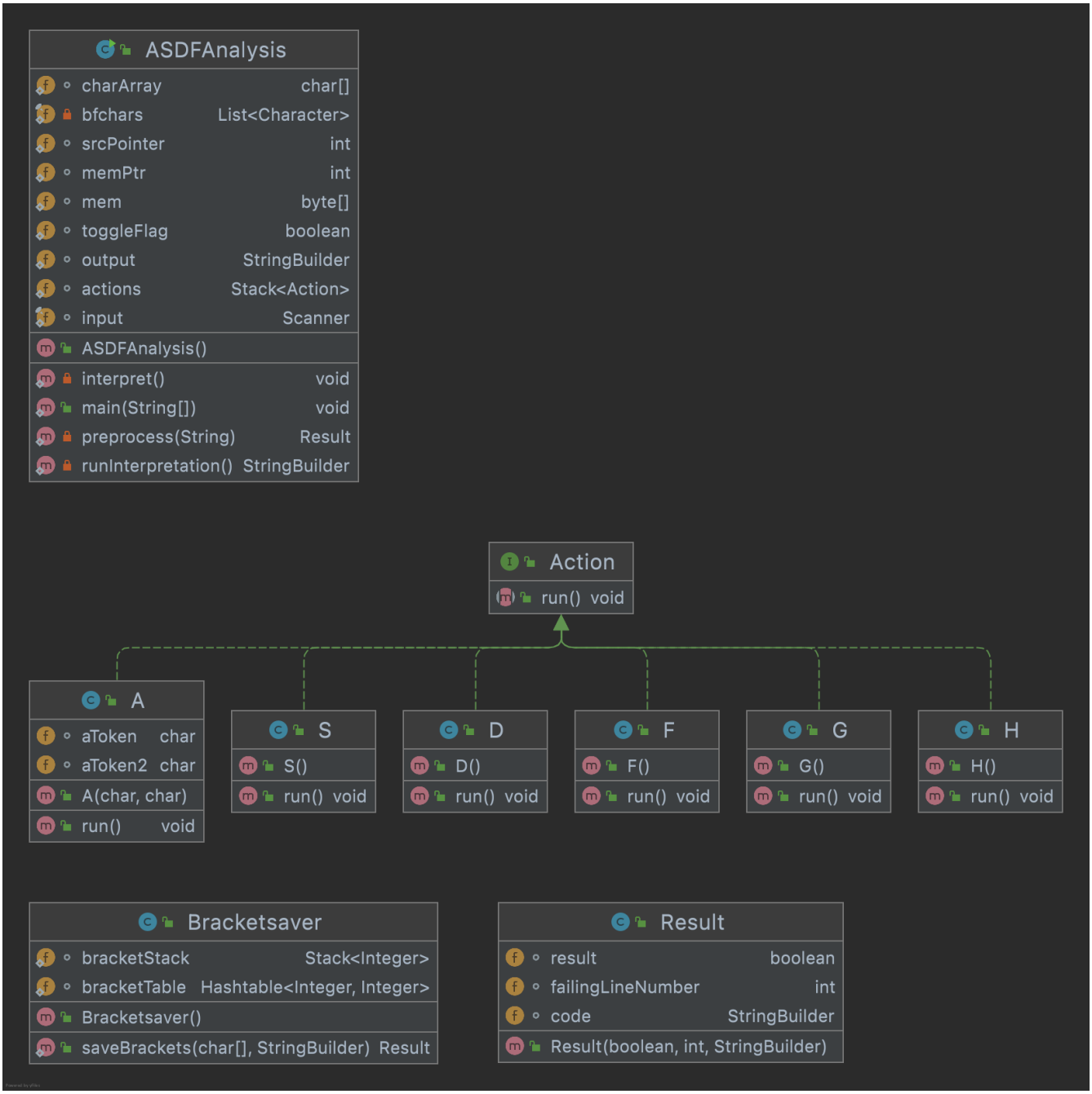
To secure valid syntax a good syntax handling is indispensable.

```
                    return new Result(false, srcPointer,new StringBuilder("Wrong f at
position" + srcPointer + "in" + code));
```

This is the handling of a error if the syntax of the brackets(d and f) isn't valid.

6.2. Level 2

6.2.1. Class Diagram



6.2.2. Action

```
public interface Action {
    public void run();
}
```

This is the central interface Action and it has a method run. Every class that implements this interface has to override this run function.

6.2.3. Why implemented with Interface

This is very good, because for example a A and a S object both have a run function which makes the code execution simpler as seen below.

```
    private static StringBuilder runInterpretation() {
        while(!actions.isEmpty()) {
            Action action = actions.pop();
            action.run();
        }
        return output;
    }
```

6.2.4. Bracketsaver and Result

```
import java.util.Hashtable;
import java.util.Stack;

public class Bracketsaver {

    static Stack<Integer> bracketStack = new Stack<>();
    static Hashtable<Integer, Integer> bracketTable = new Hashtable<>();

    public static Result saveBrackets(char[] charArray, StringBuilder code) {
        int srcPointer;
        for (srcPointer = 0; srcPointer < charArray.length; srcPointer++) {
            if (charArray[srcPointer] == 'd') {
                bracketStack.add(srcPointer);
            } else if (charArray[srcPointer] == 'f') {
                if (!bracketStack.isEmpty()) {
                    Integer previousOpenBracket = bracketStack.pop();
                    bracketTable.put(previousOpenBracket, srcPointer);
                    bracketTable.put(srcPointer, previousOpenBracket);

                } else {
                    // tag::exception[]
                    return new Result(false, srcPointer,new StringBuilder("Wrong f at
position" + srcPointer + "in" + code));
                    // end::exception[]
                }
            }
        }
        return new Result(true, 0, code);
    }
}
```

The Bracketsaver saves all d → f pairs and saves them into a Hashtable.

6.2.5. A - H

The classes A - H do the things as explained in chapter 3.

This is how the letter a is implemented. The class A needs two arguments in the constructor, to

detect if it has to write to the output.

```
public class  A implements Action {
    char aToken;
    char aToken2;
    public A(char aToken, char aToken2) {
        this.aToken = aToken;
        this.aToken2 = aToken2;
    }

    public void run() {
        if (this.aToken == 'a'){
            ASDFAnalysis.toggleFlag = true;
            ASDFAnalysis.mem[ASDFAnalysis.memPtr]++;

            if (this.aToken2 == 'a' && ASDFAnalysis.memPtr == 1) {
                // Convert integer value into char
                char aChar2 = (char) ASDFAnalysis.mem[2];
                ASDFAnalysis.output.append(aChar2);

            }
            if (this.aToken2  == 'a' && ASDFAnalysis.memPtr == 0) {
                ASDFAnalysis.mem[2] = ASDFAnalysis.input.nextByte();
            }
        }
    }
}
```

7. Runtime View

7.1. Main loop

Dot Executable: /opt/local/bin/dot
File does not exist
Cannot find Graphviz. You should try

@startuml
testdot
@enduml

or

java -jar plantuml.jar -testdot

7.2. Error occurs

Dot Executable: /opt/local/bin/dot
File does not exist
Cannot find Graphviz. You should try

@startuml
testdot
@enduml

or

java -jar plantuml.jar -testdot

7.3. Helloworld program

```
aasaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaasaahsgggsasagsaahsaaaaaaasaaahsaaasaahsgggggggggggggggggggggggggg
gggggggggggggggggggggggggggggggggggggggggggggggggggggagsaahsaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaasaahsggggggggagsaahsaaasa
ahsggggggagsaahsggggggggagsaa
```

8. Deployment View

If you want to run the application check it out from github and run ASDFAnalysis.java.

9. Cross-cutting Concepts

9.1. Versioning

The versioning is done with GitHub. The github repo link is (https://www.github.com/Lion-alt/AsdfInterpreter)

10. Architecture Decisions

10.1. Use one interface

Every new letter, for example x is a Action and has to implement from the interface Action and therefore have a run method.

10.2. Refactor the logic of each letter into its own class (f. e. A)

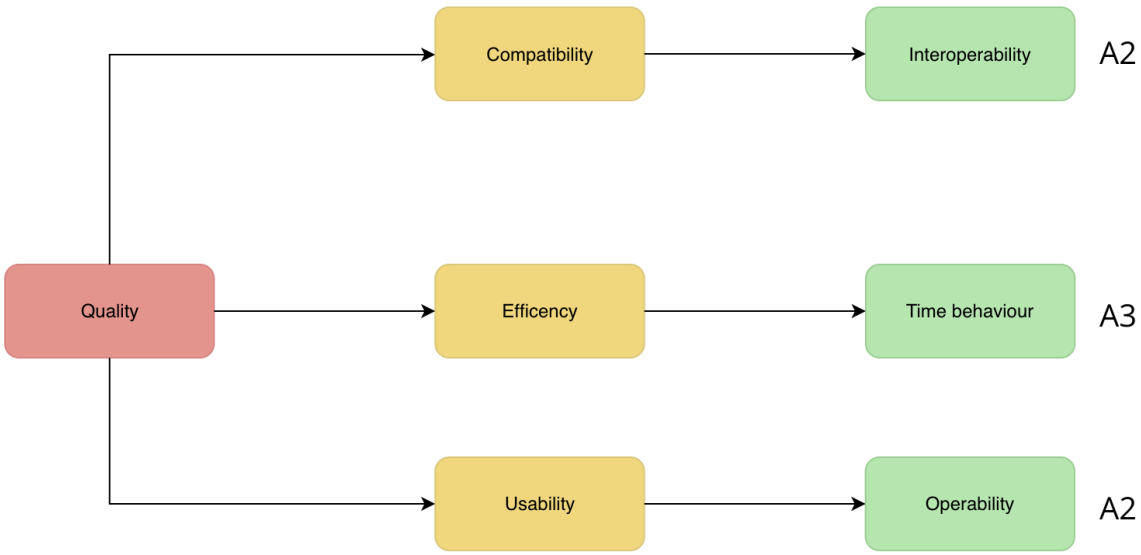
Every Letter has to have its own class and constructor arguments if needed.

10.3. Outsource the Bracketsaver into its own class

Outsource logic into respective classes. The bracketsaver has to be its own class, to handle the state of the brackettable and stack. The runtime stack could also be an extra class, but for organisational reasons I haven’t implemented it.

11. Quality Requirements

11.1. Quality Tree



11.2. Quality Scenarios

A1	**Quality concept is ok and beginner-friendly.**
A2	*Quality concept is excellent and it contains intermediate concepts...*
A3	*Concept is perfect and a joy to look at.*

12. Risks and Technical Debts

The technical risks are the following:

- User input is not validated.
- Input is not handled correctly and if too much input is provided, the program may break.
- This program is an absolute gift for every hacker, because it is touring-complete. A hacker could easily hack the deployment system, if the deployment is done with for example Heroku.

13. Overall risks

Overall risks include things like outdated java version or developer sweeping.

14. Glossary

Term	Definition
Polymorphism	*A technical concept, in object-oriented programming, where many classes implement from one interface. In this project the action interface enforces, that always the run method can be run on a action object.*
Interpreter	*A interpreter interprets code from a programming language into another programming language and executes the interpreted code. An example for this was for a long time (not anymore) Javascript. In this project the language asdf (with additional letters g and h) is interpreted into Java.*