

Soup.java

```

1 package xyz.amtstl.soup;
2
3 import java.io.BufferedReader;
12
13 /**
14  * Soup
15  * @author Alexander Christian Migala
16  */
17 public class Soup {
18     public static int lineNumber = 1;
19     public static boolean isOneLine = false;
20
21     // controllers
22     public static LogicController logic = new LogicController();
23     public static LanguageDictionary lang = new LanguageDictionary();
24     public static FlagController flags = new FlagController();
25
26     /**
27      * Main thread marshal
28      * @param args args from user
29      * @throws Exception for forced exit
30      */
31     public static void main(String args[]) throws Exception {
32         FileReader reader = null;
33         BufferedReader buff = null;
34
35         if (args[0].contains(".soup")) {
36
37             try {
38                 reader = new FileReader(System.getProperty("user.dir") + "/" +
args[0].toString());
39
40                 // pass flag
41                 try {
42                     FlagController.passFlag(args[1].toString().toLowerCase());
43                 }
44                 catch (Exception e) {
45
46                 }
47             }
48             catch (Exception ex) {
49                 System.out.println("File not found! Are you sure it is in this folder?");
50                 System.exit(0);
51             }
52             buff = new BufferedReader(reader);
53         }
54         else {
55             FlagController.execSoup(args[0]);
56
57             isOneLine = true;
58
59             try {
60                 FlagController.passFlag(args[1].toString().toLowerCase());
61             }
62             catch (Exception e) {
63
64             }

```

Soup.java

```

65     }
66
67     /*
68     * ALWAYS USE BREAKS WHEN ADDING NEW TOKENS AND FUNCTIONS
69     *
70     */
71     while (true && isOneLine == false) {
72         final String cache = buff.readLine();
73
74         try {
75             for (int i = 0; i < cache.length(); i++) {
76                 char c = cache.charAt(i);
77                 switch (c) {
78                     case '+': // add two numbers
79                         LogicController.soupAdd(i, cache);
80                         i = LogicController.index;
81                         break;
82                     case '-': // subtract two numbers
83                         LogicController.soupSubtract(i, cache);
84                         i = LogicController.index;
85                         break;
86                     case '*': // multiply two numbers
87                         LogicController.soupMultiply(i, cache);
88                         i = LogicController.index;
89                         break;
90                     case '/': // divide two numbers
91                         LogicController.soupDivide(i, cache);
92                         i = LogicController.index;
93                         break;
94                     case '^': // pow one number
95                         LogicController.soupPow(i, cache);
96                         i = LogicController.index;
97                         break;
98                     case '#': // base 10 logarithm
99                         LogicController.soupLog(i, cache);
100                        i = LogicController.index;
101                        break;
102                    case '@': // break soup
103                        System.out.println("Soup exiting with code 2 (requested per
program)");
104                        System.exit(0);
105                        break;
106                    case 'A': // area
107                        LogicController.soupArea(i, cache);
108                        i = LogicController.index;
109                        break;
110                    case '=': // basic if statement
111                        LogicController.soupIf(i, cache);
112                        i = LogicController.index;
113                        break;
114                    case 'P': // print line
115                        LogicController.soupPrint(i, cache);
116                        i = LogicController.index;
117                        break;
118                    case ';': // extension of if
119                        LogicController.soupIfDo(i, cache);
120                        i = LogicController.index;

```

Soup.java

```
121         break;
122     case ':' : // stores last result
123         LogicController.soupStoreVar(i, cache);
124         i = LogicController.index;
125         break;
126     case 'V': // gets a variable
127         LogicController.soupRetrieveVar(i, cache);
128         i = LogicController.index;
129         break;
130     case 'I': // gets var from user and stores it
131         LogicController.soupStoreUserIn(i, cache);
132         i = LogicController.index;
133         break;
134     case '$' : // trigonometric functions
135         LogicController.soupTrig(i, cache);
136         i = LogicController.index;
137         break;
138     case '|' : // absolute value
139         LogicController.soupAbs(i, cache);
140         i = LogicController.index;
141         break;
142     case '?' : // round number to int
143         LogicController.soupRound(i, cache);
144         i = LogicController.index;
145         break;
146     case '&' : // square root
147         LogicController.soupSquareRoot(i, cache);
148         i = LogicController.index;
149         break;
150     case 'R' : // random number generator
151         LogicController.soupRandomNum(i, cache);
152         i = LogicController.index;
153         break;
154     case 'H' : // html generator
155         LogicController.soupHTMLHandler(i, cache);
156         i = LogicController.index;
157         break;
158     case '~': // stores a single variable
159         LogicController.soupStoreSingle(i, cache);
160         i = LogicController.index;
161         break;
162     case '/': // comments
163         LogicController.soupComment(i, cache);
164         i = LogicController.index;
165         break;
166     case '[' : // loop
167         LogicController.soupForLoop(i, cache);
168         i = LogicController.index;
169         break;
170     case ']' :
171         break;
172     case 'W' : // while loop
173         LogicController.soupWhileLoop(i, cache);
174         i = LogicController.index;
175         break;
176     case 'D' : // for decrement
177         LogicController.soupForLoopDecre(i, cache);
```

Soup.java

```

178         i = LogicController.index;
179         break;
180     case '<' : // less than if
181         LogicController.soupIfLessThan(i, cache);
182         i = LogicController.index;
183         break;
184     case '>' : // greater than if
185         LogicController.soupIfGreaterThan(i, cache);
186         i = LogicController.index;
187         break;
188     case 'X' : // breaks loop
189         LogicController.soupBreakLoop();
190         i = LogicController.index;
191         break;
192     case 'N' : // while not
193         LogicController.soupWhileNotLoop(i, cache);
194         i = LogicController.index;
195         break;
196     case 'S' : // store a function
197         LogicController.soupStoreFunction(i, cache);
198         i = LogicController.index;
199         break;
200     case 'F' :
201         LogicController.soupGetFunction(i, cache);
202         i = LogicController.index;
203         break;
204     case '.' : // like a semicolon
205         break;
206     case ' ' : // space nullifier
207         break;
208     case ')' :
209         break;
210     case '-' :
211         break;
212     default :
213         throw new SoupSyntaxException(cache.charAt(i), i+1, lineNumber);
214     }
215
216     if (FlagController.printIndex) {
217         System.out.println("Current Index: " + String.valueOf(i));
218     }
219 }
220 } catch (NullPointerException ex) {
221     System.exit(0);
222 }
223 lineNumber++;
224 LogicController.index = 0;
225 }
226 }
227
228 /**
229  * Did the corresponding function per character
230  * @param c
231  * @param i
232  * @param cache
233  * @throws NumberFormatException
234  * @throws SoupVariableException

```

Soup.java

```

235     * @throws SoupSyntaxException
236     * @throws SoupFunctionNotDeclaredException
237     */
238     public static void parseFunc(char c, int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
239         switch (c) {
240             case '+': // add two numbers
241                 LogicController.soupAdd(i, cache);
242                 break;
243             case '-': // subtract two numbers
244                 LogicController.soupSubtract(i, cache);
245                 break;
246             case '@': // break soup
247                 System.out.println("Soup exiting with code 2 (requested per program)");
248                 System.exit(0);
249                 break;
250             case '*': // multiply two numbers
251                 LogicController.soupMultiply(i, cache);
252                 break;
253             case '%': // divide two numbers
254                 LogicController.soupDivide(i, cache);
255                 break;
256             case '^': // pow one number
257                 LogicController.soupPow(i, cache);
258                 break;
259             case '#': // basic logarithm
260                 LogicController.soupLog(i, cache);
261                 break;
262             case 'A': // area
263                 LogicController.soupArea(i, cache);
264                 break;
265             case '=': // basic if statement
266                 LogicController.soupIf(i, cache);
267                 break;
268             case ';': // extension of if
269                 LogicController.soupIfDo(i, cache);
270                 break;
271             case ':': // stores last result
272                 LogicController.soupStoreVar(i, cache);
273                 break;
274             case 'V': // gets a variable
275                 LogicController.soupRetrieveVar(i, cache);
276                 break;
277             case 'I': // gets var from user and stores it
278                 LogicController.soupStoreUserIn(i, cache);
279                 break;
280             case '$': // trig
281                 LogicController.soupTrig(i, cache);
282                 break;
283             case '|': // absolute value
284                 LogicController.soupAbs(i, cache);
285                 break;
286             case '?': // round number to int
287                 LogicController.soupRound(i, cache);
288                 break;
289             case '&': // square root
290                 LogicController.soupSquareRoot(i, cache);

```

Soup.java

```
291         break;
292     case 'R' : // random number generator
293         LogicController.soupRandomNum(i, cache);
294         break;
295     case 'H' : // html generator
296         LogicController.soupHTMLHandler(i, cache);
297         break;
298     case '~' : // stores a single variable
299         LogicController.soupStoreSingle(i, cache);
300         break;
301     case '/' : // comments
302         LogicController.soupComment(i, cache);
303         break;
304     case '[' : // loop
305         LogicController.soupForLoop(i, cache);
306         break;
307     case '.' : // like a semicolon
308         break;
309     case 'D' : // for decrement
310         LogicController.soupForLoopDecre(i, cache);
311         break;
312     case ']' :
313         break;
314     case ' ' : // space nullifier
315         break;
316     case 'P' :
317         LogicController.soupPrint(i, cache);
318         break;
319     case '<' : // less than if
320         LogicController.soupIfLessThan(i, cache);
321         break;
322     case '>' : // gretaer than if
323         LogicController.soupIfGreaterThan(i, cache);
324         break;
325     case 'W' :
326         LogicController.soupWhileLoop(i, cache);
327         break;
328     case 'N' : // while not
329         LogicController.soupWhileNotLoop(i, cache);
330         break;
331     case 'X' : // breaks loop
332         LogicController.soupBreakLoop();
333         break;
334     case ')' :
335         break;
336     case '-' :
337         break;
338     case 'S' : // store a function
339         LogicController.soupStoreFunction(i, cache);
340         break;
341     case 'F' :
342         LogicController.soupGetFunction(i, cache);
343         break;
344     default :
345         throw new SoupSyntaxException(cache.charAt(i), i+1, LineNumber);
346     }
347 }
```

Soup.java

```
348
349  /**
350   * Checks tokens
351   * @param i
352   * @param cache
353   * @param c
354   * @throws NumberFormatException
355   * @throws SoupVariableException
356   * @throws SoupSyntaxException
357   * @throws SoupFunctionNotDeclaredException
358   */
359  public static void checkToken(int i, String cache, char c) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
360      for (int e = 0; e < LanguageDictionary.LanguageTokens.size(); e++) {
361          if (cache.charAt(i) == LanguageDictionary.LanguageTokens.get(e)) {
362              parseFunc(c, i, cache);
363          }
364      }
365  }
366 }
```