```java
package xyz.amtstl.soup.logic;

import java.util.ArrayList;

/**
 * This is the main logic controller that handles all operations
 * and martials necessary controllers
 * @author Alexander C Migala
 *
 */
public class LogicController {

    /**
     * Main variable where the parsed data is
     */
    public static List<String> ns;

    public static boolean ifState = false;

    /**
     * Last result outputted by applicable functions. This gets used by the store function
     */
    public static float lastResult = 0;

    /**
     * Index cache for the main loop
     */
    public static int index;

    /**
     * Variable that locks the index
     */
    private static boolean lockIndex = false;

    /**
     * Current RandomEngine thread
     */
    private static RandomEngine rnd;

    /**
     * Breaker thread
     */
    public static boolean isBreak = false;

    /**
     * Constructor takes no args
     */
    public LogicController() {
        ns = new ArrayList<String>();
        rnd = new RandomEngine();
        VariableHandler.initiateVar();
    }

    /*
     * MATH OPERATSystem.outNS AND CONTROLS
     *
     */
```

```
68
69      /**
70       * Adds numbers.
71       * Note that the comments are depreciated ways of crunching
72       * @param i index to be passed to parser
73       * @param cache line of code from main loop
74       * @throws NumberFormatException
75       * @throws SoupVariableException
76       * @throws SoupSyntaxException
77       */
78      public static void soupAdd(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
79          ns = Parser.parse(i, cache);
80          Validator.validateNumbers(ns);
81
82          if (!lockIndex)
83          index = Parser.inx;
84
85          float out = Float.parseFloat(ns.get(0));
86          for (int e = 1; e < ns.size(); e++) {
87              out += Float.parseFloat(ns.get(e));
88          }
89
90          lastResult = out;
91          System.out.println(lastResult);
92          HTMLGen.totalOutputs.add(lastResult);
93      }
94
95      /**
96       * Subtracts numbers
97       * @param i index to be passed to parser
98       * @param cache line of code from main loop
99       * @throws NumberFormatException
100      * @throws SoupVariableException
101      * @throws SoupSyntaxException
102      */
103     public static void soupSubtract(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
104         ns = Parser.parse(i, cache);
105         Validator.validateNumbers(ns);
106
107         if (!lockIndex)
108         index = Parser.inx;
109
110         float out = Float.parseFloat(ns.get(0));
111         for (int e = 1; e < ns.size(); e++) {
112             out -= Float.parseFloat(ns.get(e));
113         }
114
115         lastResult = out;
116         System.out.println(lastResult);
117         HTMLGen.totalOutputs.add(lastResult);
118     }
119
120     /**
121      * Multiplies numbers
122      * @param i index to be passed to parser
```

```
123      * @param cache line of code from main loop
124      * @throws NumberFormatException
125      * @throws SoupVariableException
126      * @throws SoupSyntaxException
127      */
128     public static void soupMultiply(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
129         ns = Parser.parse(i, cache);
130         Validator.validateNumbers(ns);
131 
132         if (!lockIndex)
133         index = Parser.inx;
134 
135         float out = Float.parseFloat(ns.get(0));
136         for (int e = 1; e < ns.size(); e++) {
137             out *= Float.parseFloat(ns.get(e));
138         }
139 
140         lastResult = out;
141         System.out.println(lastResult);
142         HTMLGen.totalOutputs.add(lastResult);
143     }
144 
145     /**
146      * Divides numbers
147      * @param i index to be passed to parser
148      * @param cache line of code from main loop
149      * @throws NumberFormatException
150      * @throws SoupVariableException
151      * @throws SoupSyntaxException
152      */
153     public static void soupDivide(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
154         ns = Parser.parse(i, cache);
155         Validator.validateNumbers(ns);
156 
157         if (!lockIndex)
158         index = Parser.inx;
159 
160         float out = Float.parseFloat(ns.get(0));
161         for (int e = 1; e < ns.size(); e++) {
162             out /= Float.parseFloat(ns.get(e));
163         }
164 
165         lastResult = out;
166         System.out.println(lastResult);
167         HTMLGen.totalOutputs.add(lastResult);
168     }
169 
170     /**
171      * Raises numbers per exponent
172      * @param i index to be passed to parser
173      * @param cache line of code from main loop
174      * @throws NumberFormatException
175      * @throws SoupVariableException
176      * @throws SoupSyntaxException
177      */
```

```
178     public static void soupPow(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
179         ns = Parser.parse(i, cache);
180         Validator.validateNumbers(ns);
181 
182         if (!lockIndex)
183         index = Parser.inx;
184 
185         System.out.println((float)Math.pow(Float.parseFloat(ns.get(0)),
    Float.parseFloat(ns.get(1))));
186         lastResult = (float)Math.pow(Float.parseFloat(ns.get(0)),
    Float.parseFloat(ns.get(1)));
187         HTMLGen.totalOutputs.add(lastResult);
188     }
189 
190     /**
191      * Logarithms
192      * @param i index to be passed to parser
193      * @param cache line of code from main loop
194      * @throws NumberFormatException
195      * @throws SoupVariableException
196      * @throws SoupSyntaxException
197      */
198     public static void soupLog(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
199         ns = Parser.parse(i, cache);
200         Validator.validateNumbers(ns);
201 
202         if (!lockIndex)
203         index = Parser.inx;
204 
205         double ex = Double.parseDouble(ns.get(0));
206         /*double base = Double.parseDouble(ns.get(1));*/
207 
208         /*System.out.println(String.valueOf((Math.log(ex)/(Math.log(base)))));*/
209         lastResult = (float)(Math.log10(ex));
210         System.out.println(lastResult);
211         HTMLGen.totalOutputs.add(lastResult);
212     }
213 
214     /**
215      * Applies trigonometric functions
216      * @param i index to be passed to parser
217      * @param cache line of code from main loop
218      * @throws NumberFormatException
219      * @throws SoupVariableException
220      * @throws SoupSyntaxException
221      */
222     public static void soupTrig(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
223         ns = Parser.parse(i, cache);
224 
225         List<String> validation = new ArrayList<String>();
226         for (int e = 1; e < ns.size(); e++) {
227             validation.add(ns.get(e));
228         }
229         Validator.validateNumbers(validation);
```

```
230
231         if (!lockIndex)
232         index = Parser.inx;
233
234         String condition = ns.get(0);
235
236         /*
237          * Add the arc and inverse side of things
238          */
239         switch (condition) {
240         case "s" : // sine
241             System.out.println(Float.valueOf((float)
    (Math.sin(Double.parseDouble(ns.get(1))))));
242             lastResult = (float)(Math.sin(Double.parseDouble(ns.get(1))));
243             break;
244         case "c" : // cosine
245             System.out.println(Float.valueOf((float)
    (Math.cos(Double.parseDouble(ns.get(1))))));
246             lastResult = (float)(Math.cos(Double.parseDouble(ns.get(1))));
247             break;
248         case "t" : // tangent
249             System.out.println(Float.valueOf((float)
    (Math.tan(Double.parseDouble(ns.get(1))))));
250             lastResult = (float)(Math.tan(Double.parseDouble(ns.get(1))));
251             break;
252         case "arcs" : // arcsine
253             lastResult = (float)(Math.asin(Double.parseDouble(ns.get(1))));
254             System.out.println(lastResult);
255             break;
256         case "arcc" : // arccosine
257             lastResult = (float)(Math.acos(Double.parseDouble(ns.get(1))));
258             System.out.println(lastResult);
259             break;
260         case "arct" : // arctangent
261             lastResult = (float)(Math.atan(Double.parseDouble(ns.get(1))));
262             System.out.println(lastResult);
263             break;
264         default :
265             throw new SoupSyntaxException(cache.charAt(i+2), i);
266         }
267         HTMLGen.totalOutputs.add(lastResult);
268     }
269
270     /**
271      * Finds the area per the parameters
272      * @param i index to be passed to parser
273      * @param cache line of code from main loop
274      * @throws NumberFormatException
275      * @throws SoupVariableException
276      * @throws SoupSyntaxException
277      */
278     public static void soupArea(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
279         ns = Parser.parse(i, cache);
280
281         List<String> validation = new ArrayList<String>();
282         for (int e = 1; e < ns.size(); e++) {
```

```
283             validation.add(ns.get(e));
284         }
285         Validator.validateNumbers(validation);
286 
287         if (!lockIndex)
288         index = Parser.inx;
289 
290         String condition = ns.get(0);
291 
292         switch (condition) {
293         case "s" : // square
294             lastResult = Float.parseFloat(ns.get(1)) * Float.parseFloat(ns.get(2));
295             System.out.println(lastResult);
296             break;
297         case "tri" : // triangle
298             lastResult = Float.parseFloat(ns.get(1)) * Float.parseFloat(ns.get(2))/2;
299             System.out.println(lastResult);
300             break;
301         case "tra" : // trapezoid
302             float n1 = Float.parseFloat(ns.get(1));
303             float n2 = Float.parseFloat(ns.get(2));
304             float n3 = Float.parseFloat(ns.get(3));
305             lastResult = ((n1 + n2)/2) * n3;
306             System.out.println(lastResult);
307             break;
308         default :
309             throw new SoupSyntaxException(cache.charAt(i+2), i);
310         }
311         HTMLGen.totalOutputs.add(lastResult);
312     }
313 
314     /**
315      * Absolute values a number
316      * @param i index to be passed to parser
317      * @param cache line of code from main loop
318      * @throws NumberFormatException
319      * @throws SoupVariableException
320      * @throws SoupSyntaxException
321      */
322     public static void soupAbs(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
323         ns = Parser.parse(i, cache);
324         Validator.validateNumbers(ns);
325 
326         if (!lockIndex)
327         index = Parser.inx;
328 
329         lastResult = Math.abs(Float.parseFloat(ns.get(0)));
330         System.out.println(String.valueOf(lastResult));
331         HTMLGen.totalOutputs.add(lastResult);
332     }
333 
334     /**
335      * Rounds a number using Java math.round()
336      * @param i index to be passed to parser
337      * @param cache line of code from main loop
338      * @throws NumberFormatException
```

```
339      * @throws SoupVariableException
340      * @throws SoupSyntaxException
341      */
342     public static void soupRound(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
343         ns = Parser.parse(i, cache);
344         Validator.validateNumbers(ns);
345 
346         if (!lockIndex)
347         index = Parser.inx;
348 
349         lastResult = (float)Math.round(Float.valueOf(ns.get(0)));
350         System.out.println(lastResult);
351         HTMLGen.totalOutputs.add(lastResult);
352     }
353 
354     /**
355      * Square Roots a number
356      * @param i index to be passed to parser
357      * @param cache line of code from main loop
358      * @throws NumberFormatException
359      * @throws SoupVariableException
360      * @throws SoupSyntaxException
361      */
362     public static void soupSquareRoot(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
363         ns = Parser.parse(i, cache);
364         Validator.validateNumbers(ns);
365 
366         if (!lockIndex)
367         index = Parser.inx;
368 
369         lastResult = (float)Math.sqrt(Double.parseDouble(ns.get(0)));
370         System.out.println(lastResult);
371         HTMLGen.totalOutputs.add(lastResult);
372     }
373 
374     /**
375      * Will spawn a random number in lastResult between the bounds
376      * @param i index to be passed to parser
377      * @param cache line of code from main loop
378      * @throws NumberFormatException
379      * @throws SoupVariableException
380      * @throws SoupSyntaxException
381      */
382     public static void soupRandomNum(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
383         ns = Parser.parse(i, cache);
384         Validator.validateNumbers(ns);
385 
386         if (!lockIndex)
387             index = Parser.inx;
388 
389         int param1 = Integer.parseInt(ns.get(0));
390         int param2 = Integer.parseInt(ns.get(1));
391 
392         lastResult = rnd.getNumberRange(param1, param2);
```

```
393         System.out.println(lastResult);
394         HTMLGen.totalOutputs.add(lastResult);
395     }
396 
397     /*
398      * FUNCTIONALITY CONTROLS
399      *
400      */
401 
402     /**
403      * Prints some text per args
404      * @param i index to be passed to parser
405      * @param cache line of code from main loop
406      * @throws NumberFormatException
407      * @throws SoupVariableException
408      * @throws SoupSyntaxException
409      */
410     public static void soupPrint(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
411         ns = Parser.parse(i, cache);
412 
413         if (!lockIndex)
414         index = Parser.inx;
415 
416         List<String> validation = new ArrayList<String>();
417         for (int e = 1; e < ns.size(); e++) {
418             validation.add(ns.get(e));
419         }
420         Validator.validateNumbers(validation);
421 
422         switch (ns.get(1)) {
423         case "0" :
424             System.out.print(ns.get(0));
425             break;
426         case "1" :
427             System.out.println(ns.get(0));
428             break;
429         }
430     }
431 
432     /**
433      * Prints line to the user
434      * @deprecated
435      * @param i
436      * @param cache
437      * @throws NumberFormatException
438      * @throws SoupVariableException
439      * @throws SoupSyntaxException
440      */
441     public static void ~~soupPrintLine~~(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
442         ns = Parser.parse(i, cache);
443 
444         if (!lockIndex)
445         index = Parser.inx;
446         System.out.println(ns.get(0));
447     }
```

```
448
449     /**
450      * Checks two numbers and prints whether they're true or false
451      * @param i index to be passed to parser
452      * @param cache line of code from main loop
453      * @throws NumberFormatException
454      * @throws SoupVariableException
455      * @throws SoupSyntaxException
456      */
457     public static void soupIf(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
458         ns = Parser.parse(i, cache);
459         Validator.validateNumbers(ns);
460
461         if (!lockIndex)
462         index = Parser.inx;
463
464         //System.out.println(numbers[0] + " " + numbers[1]);
465
466         float n1 = Float.parseFloat(ns.get(0));
467         float n2 = Float.parseFloat(ns.get(1));
468
469         if (n1 == n2) {
470             ifState = true;
471             System.out.println("True");
472         }
473         else {
474             ifState = false;
475             System.out.println("False");
476         }
477     }
478
479     /**
480      * Checks to see if the first number is less than the second number
481      * @param i index to be passed to parser
482      * @param cache line of code from main loop
483      * @throws NumberFormatException
484      * @throws SoupVariableException
485      * @throws SoupSyntaxException
486      */
487     public static void soupIfLessThan(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
488         ns = Parser.parse(i, cache);
489         Validator.validateNumbers(ns);
490
491         if (!lockIndex)
492         index = Parser.inx;
493
494         //System.out.println(numbers[0] + " " + numbers[1]);
495
496         float n1 = Float.parseFloat(ns.get(0));
497         float n2 = Float.parseFloat(ns.get(1));
498
499         if (n1 < n2) {
500             ifState = true;
501             System.out.println("True");
502         }
```

```
503         else {
504             ifState = false;
505             System.out.println("False");
506         }
507     }
508 
509     /**
510      * Checks to see if the first number is greater than the second number
511      * @param i index to be passed to parser
512      * @param cache line of code from main loop
513      * @throws NumberFormatException
514      * @throws SoupVariableException
515      * @throws SoupSyntaxException
516      */
517     public static void soupIfGreaterThan(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
518         ns = Parser.parse(i, cache);
519         Validator.validateNumbers(ns);
520 
521         if (!lockIndex)
522         index = Parser.inx;
523 
524         //System.out.println(numbers[0] + " " + numbers[1]);
525 
526         float n1 = Float.parseFloat(ns.get(0));
527         float n2 = Float.parseFloat(ns.get(1));
528 
529         if (n1 > n2) {
530             ifState = true;
531             System.out.println("True");
532         }
533         else {
534             ifState = false;
535             System.out.println("False");
536         }
537     }
538 
539     /**
540      * This is the handler that marshals the the HTML generator
541      * @param i index to be passed to parser
542      * @param cache line of code from main loop
543      * @throws NumberFormatException
544      * @throws SoupVariableException
545      * @throws SoupSyntaxException
546      */
547     public static void soupHTMLHandler(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
548         ns = Parser.parse(i, cache);
549 
550         if (!lockIndex)
551         index = Parser.inx;
552         try {
553             HTMLGen.generateOutputDocumentation(ns.get(0), ns.get(1));
554         } catch (Exception e) {
555             // TODO Auto-generated catch block
556             e.printStackTrace();
557         }
```

```
558     }
559
560     /**
561      * Extension of if function
562      * @param i index to be passed to parser
563      * @param cache line of code from main loop
564      * @throws NumberFormatException
565      * @throws SoupVariableException
566      * @throws SoupSyntaxException
567      * @throws SoupFunctionNotDeclaredException
568      */
569     public static void soupIfDo(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
570         ns = Parser.parseInternalFunctions(i, cache);
571         index = Parser.inx;
572
573         String True = ns.get(0);
574         String False = ns.get(1);
575         lockIndex = true;
576
577         if (ifState) {
578             FunctionInterpolator.interpolateString(True);
579         }
580         else {
581             FunctionInterpolator.interpolateString(False);
582         }
583
584         lockIndex = false;
585
586         ifState = false;
587     }
588
589     /**
590      * Retrieves a variable from VariableHandler
591      * @param i index to be passed to parser
592      * @param cache line of code from main loop
593      * @throws NumberFormatException
594      * @throws SoupVariableException
595      * @throws SoupSyntaxException
596      */
597     public static void soupRetrieveVar(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
598         ns = Parser.parse(i, cache);
599         Validator.validateNumbers(ns);
600
601         if (!lockIndex)
602         index = Parser.inx;
603
604         float ret = VariableHandler.getVar(Integer.parseInt(ns.get(0)));
605         System.out.println(ret);
606     }
607
608     /**
609      * Stores a variable
610      * @param i index to be passed to parser
611      * @param cache line of code from main loop
612      * @throws NumberFormatException
```

```
613      * @throws SoupVariableException
614      * @throws SoupSyntaxException
615      */
616     public static void soupStoreVar(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
617         ns = Parser.parse(i, cache);
618         Validator.validateNumbers(ns);
619 
620         if (!lockIndex)
621         index = Parser.inx;
622 
623         VariableHandler.insertVar(lastResult, Integer.parseInt(ns.get(0)));
624     }
625 
626     /**
627      * Stores the user input
628      * @param i index to be passed to parser
629      * @param cache line of code from main loop
630      * @throws NumberFormatException
631      * @throws SoupVariableException
632      * @throws SoupSyntaxException
633      */
634     public static void soupStoreUserIn(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
635         ns = Parser.parse(i, cache);
636 
637         HTMLGen.questionStrings.add(ns.get(0));
638         List<String> validation = new ArrayList<String>();
639         for (int e = 1; e < ns.size(); e++) {
640             validation.add(ns.get(e));
641         }
642         Validator.validateNumbers(validation);
643 
644         if (!lockIndex)
645         index = Parser.inx;
646 
647         System.out.println(ns.get(0));
648         Scanner s = new Scanner(System.in);
649 
650         if (s.hasNextFloat()) {
651             lastResult = s.nextFloat();
652             VariableHandler.insertVar(lastResult, Integer.parseInt(ns.get(1)));
653         }
654 
655         else {
656             switch (s.nextLine()) {
657             case "`" :
658                 VariableHandler.insertVar(1, 100);
659                 break;
660             case "." :
661                 VariableHandler.insertVar(1, 101);
662                 break;
663             }
664         }
665     }
666 
667     /**
```

```
668      * Stores a single variable
669      * @param i index to be passed to parser
670      * @param cache line of code from main loop
671      * @throws NumberFormatException
672      * @throws SoupVariableException
673      * @throws SoupSyntaxException
674      */
675     public static void soupStoreSingle(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
676         ns = Parser.parse(i, cache);
677         Validator.validateNumbers(ns);
678 
679         if (!lockIndex)
680         index = Parser.inx;
681 
682         VariableHandler.insertVar(Float.parseFloat(ns.get(0)), Integer.valueOf(ns.get(1)));
683     }
684 
685     /**
686      * Executes a new for loop on main thread
687      * @param i index to be passed to parser
688      * @param cache line of code from main loop
689      * @throws NumberFormatException
690      * @throws SoupVariableException
691      * @throws SoupSyntaxException
692      * @throws SoupFunctionNotDeclaredException
693      */
694     public static void soupForLoop(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
695         ns = Parser.parse(i, cache);
696 
697         if (!lockIndex)
698         index = Parser.inx + 1;
699         Looper.execNewForLoop((int)Integer.valueOf((int) Float.parseFloat(ns.get(0))),
    (int)Integer.valueOf((int) Float.parseFloat(ns.get(1))), cache, " ");
700     }
701 
702     /**
703      * Decrementing For Loop
704      * @param i index to be passed to parser
705      * @param cache line of code from main loop
706      * @throws NumberFormatException
707      * @throws SoupVariableException
708      * @throws SoupSyntaxException
709      * @throws SoupFunctionNotDeclaredException
710      */
711     public static void soupForLoopDecre(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
712         ns = Parser.parse(i, cache);
713 
714         if (!lockIndex)
715             index = Parser.inx + 1;
716 
717 
718         if (isBreak) {
719             isBreak = false;
720         }
```

```
721         Looper.execNewForLoopDecre((int)Integer.valueOf((int) Float.parseFloat(ns.get(0))),
    (int)Integer.valueOf((int) Float.parseFloat(ns.get(1))), cache, " ");
722     }
723
724     /**
725      * Does a while loop
726      * @param i index to be passed to parser
727      * @param cache line of code from main loop
728      * @throws SoupSyntaxException
729      * @throws NumberFormatException
730      * @throws SoupVariableException
731      * @throws SoupFunctionNotDeclaredException
732      */
733     public static void soupWhileLoop(int i, String cache) throws SoupSyntaxException,
    NumberFormatException, SoupVariableException, SoupFunctionNotDeclaredException {
734         ns = Parser.parse(i, cache);
735
736         if (!lockIndex)
737         index = Parser.inx;
738
739         Looper.execNewWhileLoop(cache);
740     }
741
742     /**
743      * Does a while not loop
744      * @param i index to be passed to parser
745      * @param cache line of code from main loop
746      * @throws SoupSyntaxException
747      * @throws NumberFormatException
748      * @throws SoupVariableException
749      * @throws SoupFunctionNotDeclaredException
750      */
751     public static void soupWhileNotLoop(int i, String cache) throws SoupSyntaxException,
    NumberFormatException, SoupVariableException, SoupFunctionNotDeclaredException {
752         ns = Parser.parse(i, cache);
753
754         if (!lockIndex)
755         index = Parser.inx;
756
757         Looper.execNewWhileNotLoop(cache);
758     }
759
760     /**
761      * Refreshes Numbers
762      * @deprecated
763      * @param i index to be passed to parser
764      * @param cache line of code from main loop
765      * @throws NumberFormatException
766      * @throws SoupVariableException
767      * @throws SoupSyntaxException
768      */
769     public static void soupRefreshNumbers(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
770         ns = Parser.parse(i, cache);
771         //Validator.validateNumbers(ns);
772         if (!lockIndex)
773         index = Parser.inx;
```

```
774     }
775
776     /**
777      * Stores a function
778      * @param i index to be passed to parser
779      * @param cache line of code from main loop
780      * @throws NumberFormatException
781      * @throws SoupVariableException
782      * @throws SoupSyntaxException
783      */
784     public static void soupStoreFunction(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
785         ns = Parser.parseInternalFunctions(i, cache);
786
787         if (!lockIndex) {
788             index = Parser.inx;
789         }
790
791         String function = ns.get(0);
792         int point = Integer.valueOf(ns.get(1));
793
794         VariableHandler.stringVars.set(point, function);
795     }
796
797     /**
798      * Gets a function and executes it
799      * @param i index to be passed to parser
800      * @param cache line of code from main loop
801      * @throws NumberFormatException
802      * @throws SoupVariableException
803      * @throws SoupSyntaxException
804      * @throws SoupFunctionNotDeclaredException
805      */
806     public static void soupGetFunction(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
807         ns = Parser.parse(i, cache);
808
809         if (!lockIndex) {
810             index = Parser.inx;
811         }
812
813         int point = Integer.valueOf(ns.get(0));
814
815         if (VariableHandler.stringVars.get(point) == "") {
816             throw new SoupFunctionNotDeclaredException(point);
817         }
818         else {
819             FunctionInterpolator.interpolateString(VariableHandler.stringVars.get(point));
820         }
821     }
822
823     /**
824      * Breaks a current loop
825      */
826     public static void soupBreakLoop() {
827         if (isBreak == false) {
828             isBreak = true;
```

```
829             index += 1;
830         }
831         else {
832             isBreak = false;
833         }
834     }
835 
836     public static void setBreak(boolean condition) {
837         isBreak = condition;
838     }
839 
840     /**
841      * Parses comments
842      * @param i index to be passed to parser
843      * @param cache line of code from main loop
844      * @throws NumberFormatException
845      * @throws SoupVariableException
846      */
847     public static void soupComment(int i, String cache) throws NumberFormatException,
    SoupVariableException {
848         index = cache.length();
849     }
850 }
```