```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 {

    /**
     * Parser Variable
     */
    public static Parser p;

    /**
     * VariableHandler global var
     */
    public static VariableHandler v;

    /**
     * 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
     */
    private static float lastResult = 0;

    /**
     * Index cache for the main loop
     */
    private 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
     */
```

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

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

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

```
232      * @throws SoupVariableException
233      * @throws SoupSyntaxException
234      */
235     public void soupTrig(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
236         ns = p.parse(i, cache);
237 
238         List<String> validation = new ArrayList<String>();
239         for (int e = 1; e < ns.size(); e++) {
240             validation.add(ns.get(e));
241         }
242         Validator.validateNumbers(validation);
243 
244         if (!lockIndex)
245         index = p.getIndex();
246 
247         String condition = ns.get(0);
248 
249         /*
250          * Add the arc and inverse side of things
251          */
252         switch (condition) {
253         case "s" : // sine
254             System.out.println(Float.valueOf((float)
    (Math.sin(Double.parseDouble(ns.get(1))))));
255             lastResult = (float)(Math.sin(Double.parseDouble(ns.get(1))));
256             break;
257         case "c" : // cosine
258             System.out.println(Float.valueOf((float)
    (Math.cos(Double.parseDouble(ns.get(1))))));
259             lastResult = (float)(Math.cos(Double.parseDouble(ns.get(1))));
260             break;
261         case "t" : // tangent
262             System.out.println(Float.valueOf((float)
    (Math.tan(Double.parseDouble(ns.get(1))))));
263             lastResult = (float)(Math.tan(Double.parseDouble(ns.get(1))));
264             break;
265         case "arcs" : // arcsine
266             lastResult = (float)(Math.asin(Double.parseDouble(ns.get(1))));
267             System.out.println(lastResult);
268             break;
269         case "arcc" : // arccosine
270             lastResult = (float)(Math.acos(Double.parseDouble(ns.get(1))));
271             System.out.println(lastResult);
272             break;
273         case "arct" : // arctangent
274             lastResult = (float)(Math.atan(Double.parseDouble(ns.get(1))));
275             System.out.println(lastResult);
276             break;
277         default :
278             throw new SoupSyntaxException(cache.charAt(i+2), i);
279         }
280         HTMLGen.getTotalOutputs().add(lastResult);
281     }
282 
283     /**
284      * Finds the area per the parameters
```

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

```
340         index = p.getIndex();
341 
342         lastResult = Math.abs(Float.parseFloat(ns.get(0)));
343         System.out.println(String.valueOf(lastResult));
344         HTMLGen.getTotalOutputs().add(lastResult);
345     }
346 
347     /**
348      * Rounds a number using Java math.round()
349      * @param i index to be passed to parser
350      * @param cache line of code from main loop
351      * @throws NumberFormatException
352      * @throws SoupVariableException
353      * @throws SoupSyntaxException
354      */
355     public void soupRound(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
356         ns = p.parse(i, cache);
357         Validator.validateNumbers(ns);
358 
359         if (!lockIndex)
360         index = p.getIndex();
361 
362         lastResult = (float)Math.round(Float.valueOf(ns.get(0)));
363         System.out.println(lastResult);
364         HTMLGen.getTotalOutputs().add(lastResult);
365     }
366 
367     /**
368      * Square Roots a number
369      * @param i index to be passed to parser
370      * @param cache line of code from main loop
371      * @throws NumberFormatException
372      * @throws SoupVariableException
373      * @throws SoupSyntaxException
374      */
375     public void soupSquareRoot(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
376         ns = p.parse(i, cache);
377         Validator.validateNumbers(ns);
378 
379         if (!lockIndex)
380         index = p.getIndex();
381 
382         lastResult = (float)Math.sqrt(Double.parseDouble(ns.get(0)));
383         System.out.println(lastResult);
384         HTMLGen.getTotalOutputs().add(lastResult);
385     }
386 
387     /**
388      * Will spawn a random number in lastResult between the bounds
389      * @param i index to be passed to parser
390      * @param cache line of code from main loop
391      * @throws NumberFormatException
392      * @throws SoupVariableException
393      * @throws SoupSyntaxException
394      */
```

```
395     public void soupRandomNum(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
396         ns = p.parse(i, cache);
397         Validator.validateNumbers(ns);
398 
399         if (!lockIndex)
400             index = p.getIndex();
401 
402         int param1 = Integer.parseInt(ns.get(0));
403         int param2 = Integer.parseInt(ns.get(1));
404 
405         lastResult = rnd.getNumberRange(param1, param2);
406         System.out.println(lastResult);
407         HTMLGen.getTotalOutputs().add(lastResult);
408     }
409 
410     /*
411      * FUNCTSystem.outNALITY CONTROLS
412      *
413      */
414 
415     /**
416      * Prints some text per args
417      * @param i index to be passed to parser
418      * @param cache line of code from main loop
419      * @throws NumberFormatException
420      * @throws SoupVariableException
421      * @throws SoupSyntaxException
422      */
423     public void soupPrint(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
424         ns = p.parse(i, cache);
425 
426         if (!lockIndex)
427         index = p.getIndex();
428 
429         List<String> validation = new ArrayList<String>();
430         for (int e = 1; e < ns.size(); e++) {
431             validation.add(ns.get(e));
432         }
433         Validator.validateNumbers(validation);
434 
435         switch (ns.get(1)) {
436         case "0" :
437             System.out.print(ns.get(0));
438             break;
439         case "1" :
440             System.out.println(ns.get(0));
441             break;
442         }
443     }
444 
445     /**
446      * Prints line to the user
447      * @deprecated
448      * @param i
449      * @param cache
```

```
450      * @throws NumberFormatException
451      * @throws SoupVariableException
452      * @throws SoupSyntaxException
453      */
454     public void ~~soupPrintLine~~(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
455         ns = p.parse(i, cache);
456 
457         if (!lockIndex)
458         index = p.getIndex();
459         System.out.println(ns.get(0));
460     }
461 
462     /**
463      * Checks two numbers and prints whether they're true or false
464      * @param i index to be passed to parser
465      * @param cache line of code from main loop
466      * @throws NumberFormatException
467      * @throws SoupVariableException
468      * @throws SoupSyntaxException
469      */
470     public void soupIf(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
471         ns = p.parse(i, cache);
472         Validator.validateNumbers(ns);
473 
474         if (!lockIndex)
475         index = p.getIndex();
476 
477         //System.out.println(numbers[0] + " " + numbers[1]);
478 
479         float n1 = Float.parseFloat(ns.get(0));
480         float n2 = Float.parseFloat(ns.get(1));
481 
482         if (n1 == n2) {
483             ifState = true;
484             System.out.println("True");
485         }
486         else {
487             ifState = false;
488             System.out.println("False");
489         }
490     }
491 
492     /**
493      * Checks to see if the first number is less than the second number
494      * @param i index to be passed to parser
495      * @param cache line of code from main loop
496      * @throws NumberFormatException
497      * @throws SoupVariableException
498      * @throws SoupSyntaxException
499      */
500     public void soupIfLessThan(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
501         ns = p.parse(i, cache);
502         Validator.validateNumbers(ns);
503 
```

```
504         if (!lockIndex)
505         index = p.getIndex();
506 
507         //System.out.println(numbers[0] + " " + numbers[1]);
508 
509         float n1 = Float.parseFloat(ns.get(0));
510         float n2 = Float.parseFloat(ns.get(1));
511 
512         if (n1 < n2) {
513             ifState = true;
514             System.out.println("True");
515         }
516         else {
517             ifState = false;
518             System.out.println("False");
519         }
520     }
521 
522     /**
523      * Checks to see if the first number is greater than the second number
524      * @param i index to be passed to parser
525      * @param cache line of code from main loop
526      * @throws NumberFormatException
527      * @throws SoupVariableException
528      * @throws SoupSyntaxException
529      */
530     public void soupIfGreaterThan(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
531         ns = p.parse(i, cache);
532         Validator.validateNumbers(ns);
533 
534         if (!lockIndex)
535         index = p.getIndex();
536 
537         //System.out.println(numbers[0] + " " + numbers[1]);
538 
539         float n1 = Float.parseFloat(ns.get(0));
540         float n2 = Float.parseFloat(ns.get(1));
541 
542         if (n1 > n2) {
543             ifState = true;
544             System.out.println("True");
545         }
546         else {
547             ifState = false;
548             System.out.println("False");
549         }
550     }
551 
552     /**
553      * This is the handler that marshals the the HTML generator
554      * @param i index to be passed to parser
555      * @param cache line of code from main loop
556      * @throws NumberFormatException
557      * @throws SoupVariableException
558      * @throws SoupSyntaxException
559      */
```

```
560     public void soupHTMLHandler(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
561         ns = p.parse(i, cache);
562 
563         if (!lockIndex)
564         index = p.getIndex();
565         try {
566             HTMLGen.generateOutputDocumentation(ns.get(0), ns.get(1));
567         } catch (Exception e) {
568             // TODO Auto-generated catch block
569             e.printStackTrace();
570         }
571     }
572 
573     /**
574      * Extension of if function
575      * @param i index to be passed to parser
576      * @param cache line of code from main loop
577      * @throws NumberFormatException
578      * @throws SoupVariableException
579      * @throws SoupSyntaxException
580      * @throws SoupFunctionNotDeclaredException
581      */
582     public void soupIfDo(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
583         ns = p.parseInternalFunctions(i, cache);
584         index = p.getIndex();
585 
586         String True = ns.get(0);
587         String False = ns.get(1);
588         lockIndex = true;
589 
590         if (ifState) {
591             FunctionInterpolator.interpolateString(True);
592         }
593         else {
594             FunctionInterpolator.interpolateString(False);
595         }
596 
597         lockIndex = false;
598 
599         ifState = false;
600     }
601 
602     /**
603      * Retrieves a variable from VariableHandler
604      * @param i index to be passed to parser
605      * @param cache line of code from main loop
606      * @throws NumberFormatException
607      * @throws SoupVariableException
608      * @throws SoupSyntaxException
609      */
610     public void soupRetrieveVar(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
611         ns = p.parse(i, cache);
612         Validator.validateNumbers(ns);
613 
```

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

```
669             switch (s.nextLine()) {
670             case "`" :
671                 v.insertVar(1, 100);
672                 break;
673             case "." :
674                 v.insertVar(1, 101);
675                 break;
676             }
677         }
678     }
679 
680     /**
681      * Stores a single variable
682      * @param i index to be passed to parser
683      * @param cache line of code from main loop
684      * @throws NumberFormatException
685      * @throws SoupVariableException
686      * @throws SoupSyntaxException
687      */
688     public void soupStoreSingle(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException {
689         ns = p.parse(i, cache);
690         Validator.validateNumbers(ns);
691 
692         if (!lockIndex)
693         index = p.getIndex();
694 
695         v.insertVar(Float.parseFloat(ns.get(0)), Integer.valueOf(ns.get(1)));
696     }
697 
698     /**
699      * Executes a new for loop on main thread
700      * @param i index to be passed to parser
701      * @param cache line of code from main loop
702      * @throws NumberFormatException
703      * @throws SoupVariableException
704      * @throws SoupSyntaxException
705      * @throws SoupFunctionNotDeclaredException
706      */
707     public void soupForLoop(int i, String cache) throws NumberFormatException,
    SoupVariableException, SoupSyntaxException, SoupFunctionNotDeclaredException {
708         ns = p.parse(i, cache);
709 
710         if (!lockIndex)
711         index = p.getIndex() + 1;
712         Looper.execNewForLoop((int)Integer.valueOf((int) Float.parseFloat(ns.get(0))),
    (int)Integer.valueOf((int) Float.parseFloat(ns.get(1))), cache, " ");
713     }
714 
715     /**
716      * Decrementing For Loop
717      * @param i index to be passed to parser
718      * @param cache line of code from main loop
719      * @throws NumberFormatException
720      * @throws SoupVariableException
721      * @throws SoupSyntaxException
722      * @throws SoupFunctionNotDeclaredException
```

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

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

```
830         }
831         else {
832             FunctionInterpolator.interpolateString(v.getStrings().get(point));
833         }
834     }
835 
836     /**
837      * Breaks a current loop
838      */
839     public void soupBreakLoop() {
840         if (isBreak == false) {
841             isBreak = true;
842             index += 1;
843         }
844         else {
845             isBreak = false;
846         }
847     }
848 
849     public static void setBreak(boolean condition) {
850         isBreak = condition;
851     }
852 
853     /**
854      * Parses comments
855      * @param i index to be passed to parser
856      * @param cache line of code from main loop
857      * @throws NumberFormatException
858      * @throws SoupVariableException
859      */
860     public void soupComment(int i, String cache) throws NumberFormatException,
    SoupVariableException {
861         index = cache.length();
862     }
863 
864     /**
865      * Gets the current index
866      * @return the current index
867      */
868     public int getIndex() {
869         return index;
870     }
871 
872     /**
873      * Sets the current index
874      * @param newIndex the new index
875      */
876     public void setIndex(int newIndex) {
877         index = newIndex;
878     }
879 
880     /**
881      * Gets the last result
882      * @return the last result variable
883      */
884     public static float getLastResult() {
885         return lastResult;
```

```
886     }
887 }
```