```java
package aston.gui;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import aston.simulator.*;

/**
 * The Simulation as a Graphical Interface. This first creates the Parameter GUI that sets
 * the information, from probability to the number of tills and pumps. This runs a
 * simulation b ased on the given informaiton from the Parameter GUI
 *
 * @author Kelvin M.
 * @author Tristan P.
 *
 * @version 19/04/2017
 *
 */
public class PetrolGUI {

    // JFrame for the Parameter Setting
    private JFrame mainParameterFrame;

    private JSlider pSlider;            //Probability of P
```

```
30      private JSlider qSlider;            //Probability of Q
31      private JSlider pumpSlider;         //Number of Pumps
32      private JSlider tillSlider;         //Number of Tills
33      private JTextField stepField;       //Simulation Steps
34      private JTextField priceField;      //Price per Gallon
35      private JCheckBox truckCheck;       //L1: With/Without Trucks
36
37      // JFrame for the Actual Simulation
38      private JFrame simulationFrame;
39
40      private JLabel ticksStep;           //Simulation Step Tracker
41      private JLabel simMoney;            //Overall Total Amount of Money from Current
    Simulation
42      private JLabel simLoss;             //Overall Total Amount of Money Loss from
    Current Simulation
43
44      // Array of JTextFields instead of declaring each JTextField one by one
45      private JTextField pumpFields[];    //Array of JTextFields for Pumps
46      // Text Areas
47      private JTextArea tillFields[];
48
49      // Class Declaration for Simulation
50      private Simulator s;
51
52      /**
53       * Petrol GUI Constructor.<br>
54       * The GUI Simulation of the Petrol Pump Simulator to set the Parameters
55       */
56      public PetrolGUI() {
57
```

```
58          // Instantiate the Simulator
59          s = new Simulator(this);
60
61          // Step 1: Create the components
62          JLabel title = new JLabel();
63
64          JLabel label1 = new JLabel();
65          pSlider = new JSlider();
66          JLabel slidePNum = new JLabel();
67
68          JLabel label2 = new JLabel();
69          qSlider = new JSlider();
70          JLabel slideQNum = new JLabel();
71
72          JLabel label5 = new JLabel();
73          pumpSlider = new JSlider();
74          JLabel pumpSlideNum = new JLabel();
75
76          JLabel label6 = new JLabel();
77          tillSlider = new JSlider();
78          JLabel tillSlideNum = new JLabel();
79
80          JLabel label3 = new JLabel();
81          stepField = new JTextField();
82
83          JLabel label4 = new JLabel();
84          priceField = new JTextField();
85
86          JButton runButton = new JButton();
87          JButton quitButton = new JButton();
```

```
 88
 89          JLabel label7 = new JLabel();
 90          truckCheck = new JCheckBox();
 91
 92          // Step 2: Set the properties of the components
 93          title.setText("Petrol Simulator Parameters");
 94          label1.setText("Probability of p:");
 95          label2.setText("Probability of q:");
 96          label3.setText("Simulation Time (ticks):");
 97          label4.setText("Price per Gallon:");
 98          label5.setText("Number of Pumps:");
 99          label6.setText("Number of Tills:");
100          label7.setText("With/Without Trucks:");
101
102              // probabilityQ slider
103          qSlider.setMinimum(1);
104          qSlider.setMaximum(5);
105          qSlider.setValue(0);
106          qSlider.setMajorTickSpacing(1);
107          qSlider.setToolTipText("Probability of Q");
108          qSlider.setPaintTicks(true);
109          slideQNum.setText("0.01");
110
111              // probabilityP slider
112          pSlider.setMinimum(1);
113          pSlider.setMaximum(5);
114          pSlider.setValue(0);
115          pSlider.setMajorTickSpacing(1);
116          pSlider.setToolTipText("Probability of P");
117          pSlider.setPaintTicks(true);
```

```
118         slidePNum.setText("0.01");
119 
120             // numPump slider
121         pumpSlider.setMinimum(0);
122         pumpSlider.setMaximum(2);
123         pumpSlider.setValue(0);
124         pumpSlider.setMajorTickSpacing(1);
125         pumpSlider.setPaintTicks(true);
126         pumpSlider.setToolTipText("Number of Pumps");
127         pumpSlideNum.setText("1");
128 
129             // numTill Slider
130         tillSlider.setMinimum(0);
131         tillSlider.setMaximum(2);
132         tillSlider.setValue(0);
133         tillSlider.setMajorTickSpacing(1);
134         tillSlider.setPaintTicks(true);
135         tillSlider.setToolTipText("Number of Tills");
136         tillSlideNum.setText("1");
137 
138             // Step Field
139         stepField.setText("1440");
140 
141             // Price Field
142         priceField.setText("1.20");
143 
144             // Button
145         runButton.setText("Run Simulation");
146         runButton.setToolTipText("Start the Simulation");
147         quitButton.setText("Quit Simulation");
```

```
148         quitButton.setToolTipText("Quit the Program");
149 
150             // Checkbox
151         truckCheck.setSelected(true);
152 
153         // Step 3: Create containers to hold the components
154         mainParameterFrame = new JFrame("Petrol Station Simulation Parameters");
155 
    mainParameterFrame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
156 
157         // Step 4: Specify LayoutManagers
158         mainParameterFrame.setLayout(new BorderLayout());
159         ((JPanel) mainParameterFrame.getContentPane()).setBorder(new EmptyBorder(6, 6,
    6, 6));
160 
161         JPanel pPanel = new JPanel();
162         pPanel.setLayout(new BorderLayout());
163         pPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
164 
165         JPanel qPanel = new JPanel();
166         qPanel.setLayout(new BorderLayout());
167         qPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
168 
169         JPanel pumpPanel = new JPanel();
170         pumpPanel.setLayout(new BorderLayout());
171         pumpPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
172 
173         JPanel tillPanel = new JPanel();
174         tillPanel.setLayout(new BorderLayout());
175         tillPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
```

```
176
177         JPanel stepPanel = new JPanel();
178         stepPanel.setLayout(new BorderLayout());
179         stepPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
180
181         JPanel pricePanel = new JPanel();
182         pricePanel.setLayout(new BorderLayout());
183         pricePanel.setBorder(new EmptyBorder(6, 6, 6, 6));
184
185         JPanel truckPanel = new JPanel();
186         truckPanel.setLayout(new BorderLayout());
187         truckPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
188
189         JPanel paramPanel = new JPanel();
190         paramPanel.setLayout(new BorderLayout());
191         paramPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
192
193         JPanel paramTopPanel = new JPanel();
194         paramTopPanel.setLayout(new BorderLayout());
195         paramTopPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
196
197         JPanel paramBotPanel = new JPanel();
198         paramBotPanel.setLayout(new BorderLayout());
199         paramBotPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
200
201         JPanel buttonPanel = new JPanel();
202         buttonPanel.setLayout(new BorderLayout());
203         buttonPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
204
205         // Step 5: Add components to containers
```

```
206         pPanel.add(label1, BorderLayout.WEST);
207         pPanel.add(pSlider, BorderLayout.CENTER);
208         pPanel.add(slidePNum, BorderLayout.EAST);
209 
210         qPanel.add(label2, BorderLayout.WEST);
211         qPanel.add(qSlider, BorderLayout.CENTER);
212         qPanel.add(slideQNum, BorderLayout.EAST);
213 
214         pumpPanel.add(label5, BorderLayout.WEST);
215         pumpPanel.add(pumpSlider, BorderLayout.CENTER);
216         pumpPanel.add(pumpSlideNum, BorderLayout.EAST);
217 
218         stepPanel.add(label3, BorderLayout.WEST);
219         stepPanel.add(stepField, BorderLayout.CENTER);
220 
221         pricePanel.add(label4, BorderLayout.WEST);
222         pricePanel.add(priceField, BorderLayout.CENTER);
223 
224         tillPanel.add(label6, BorderLayout.WEST);
225         tillPanel.add(tillSlider, BorderLayout.CENTER);
226         tillPanel.add(tillSlideNum, BorderLayout.EAST);
227 
228         truckPanel.add(label7, BorderLayout.WEST);
229         truckPanel.add(truckCheck, BorderLayout.CENTER);
230 
231         paramTopPanel.add(pPanel, BorderLayout.NORTH);
232         paramTopPanel.add(qPanel, BorderLayout.CENTER);
233         paramTopPanel.add(pumpPanel, BorderLayout.SOUTH);
234 
235         paramBotPanel.add(tillPanel, BorderLayout.NORTH);
```

```
236         paramBotPanel.add(stepPanel, BorderLayout.CENTER);
237         paramBotPanel.add(pricePanel, BorderLayout.SOUTH);
238 
239         buttonPanel.add(runButton, BorderLayout.WEST);
240         buttonPanel.add(quitButton, BorderLayout.EAST);
241 
242         paramPanel.add(paramTopPanel, BorderLayout.NORTH);
243         paramPanel.add(paramBotPanel, BorderLayout.CENTER);
244         paramPanel.add(truckPanel, BorderLayout.SOUTH);
245 
246         mainParameterFrame.add(title, BorderLayout.NORTH);
247         mainParameterFrame.add(paramPanel, BorderLayout.CENTER);
248         mainParameterFrame.add(buttonPanel, BorderLayout.SOUTH);
249 
250         // Step 6: Arrange to handle events in the user interface
251             //When the red close button is clicked
252         mainParameterFrame.addWindowListener(new WindowAdapter() {
253             public void windowClosing(WindowEvent e) {
254                 exitApp();
255             }
256         });
257 
258             //When the Quit Button is clicked
259         quitButton.addActionListener(new ActionListener() {
260             public void actionPerformed(ActionEvent e) {
261                 exitApp();
262             }
263         });
264 
265             //When the Run Button is clicked
```

```
266         runButton.addActionListener(new ActionListener() {
267             public void actionPerformed(ActionEvent e) {
268                 //If the Truck is Selected
269                 if(truckCheck.isSelected() == true){
270                     //Run the Simulation WITH TRUCK
271                     newSimulation();
272                 } else {
273                     //Run the Simulation WITH OUT TRUCK
274                     newSimulation();
275                 }
276             }
277         });
278 
279             //When the Probability of P is moved
280         pSlider.addChangeListener(new ChangeListener() {
281             public void stateChanged(ChangeEvent e) {
282                 int value = ((JSlider) e.getSource()).getValue();
283                 slidePNum.setText("0.0" + value + "");
284 
285             }
286         });
287 
288             //When the Probability of Q is moved
289         qSlider.addChangeListener(new ChangeListener() {
290             public void stateChanged(ChangeEvent e) {
291                 int value = ((JSlider) e.getSource()).getValue();
292                 slideQNum.setText("0.0" + value + "");
293 
294             }
295         });
```

```
296
297             //When the Number of Pump Slider is moved
298         pumpSlider.addChangeListener(new ChangeListener() {
299             public void stateChanged(ChangeEvent e) {
300                 int value = ((JSlider) e.getSource()).getValue();
301                 int present = (int) Math.pow(2, value);
302                 pumpSlideNum.setText(present + "");
303
304             }
305         });
306
307             //When the Number of Tills Slider is moved
308         tillSlider.addChangeListener(new ChangeListener() {
309             public void stateChanged(ChangeEvent e) {
310                 int value = ((JSlider) e.getSource()).getValue();
311                 int present = (int) Math.pow(2, value);
312                 tillSlideNum.setText(present + "");
313
314             }
315         });
316
317         // Step 7: Display the GUI
318         mainParameterFrame.pack();
319         mainParameterFrame.setVisible(true);
320     }
321
322     /**
323      * GUI Simulation presenting the Pump's/Till's queue.<br>
324      * This is created based on the parameters given from the Main GUI Frame
325      */
```

```
326     private void newSimulation() {
327 
328         //Set up JTextField for output
329         initJTxtFld();
330 
331         // Step 1: Create the components
332         JButton stopButton = new JButton();
333         JLabel titleLog = new JLabel();
334 
335         ticksStep = new JLabel();
336         simMoney = new JLabel();
337         simLoss = new JLabel();
338 
339         //Pumps
340             //Row 1
341         JLabel pumpLabel1 = new JLabel();
342 
343         pumpFields[0] = new JTextField(12);
344         pumpFields[0].setEditable(false);
345         pumpFields[0].setBackground(new Color(218, 247, 166));
346         pumpFields[1] = new JTextField(12);
347         pumpFields[1].setEditable(false);
348         pumpFields[2] = new JTextField(12);
349         pumpFields[2].setEditable(false);
350         pumpFields[3] = new JTextField(12);
351         pumpFields[3].setEditable(false);
352 
353             //Row 2
354         JLabel pumpLabel2 = new JLabel();
355 
```

```
356         pumpFields[4] = new JTextField(12);
357         pumpFields[4].setEditable(false);
358         pumpFields[4].setBackground(new Color(218, 247, 166));
359         pumpFields[5] = new JTextField(12);
360         pumpFields[5].setEditable(false);
361         pumpFields[6] = new JTextField(12);
362         pumpFields[6].setEditable(false);
363         pumpFields[7] = new JTextField(12);
364         pumpFields[7].setEditable(false);
365 
366             //Row 3
367         JLabel pumpLabel3 = new JLabel();
368 
369         pumpFields[8] = new JTextField(12);
370         pumpFields[8].setEditable(false);
371         pumpFields[8].setBackground(new Color(218, 247, 166));
372         pumpFields[9] = new JTextField(12);
373         pumpFields[9].setEditable(false);
374         pumpFields[10] = new JTextField(12);
375         pumpFields[10].setEditable(false);
376         pumpFields[11] = new JTextField(12);
377         pumpFields[11].setEditable(false);
378 
379             //Row 4
380         JLabel pumpLabel4 = new JLabel();
381 
382         pumpFields[12] = new JTextField(12);
383         pumpFields[12].setEditable(false);
384         pumpFields[12].setBackground(new Color(218, 247, 166));
385         pumpFields[13] = new JTextField(12);
```

```
386         pumpFields[13].setEditable(false);
387         pumpFields[14] = new JTextField(12);
388         pumpFields[14].setEditable(false);
389         pumpFields[15] = new JTextField(12);
390         pumpFields[15].setEditable(false);
391 
392         //Tills
393             //Row 1
394         JLabel tillLabel1 = new JLabel();
395 
396         tillFields[0] = new JTextArea(6,12);
397         tillFields[0].setEditable(false);
398         tillFields[0].getPreferredScrollableViewportSize();
399 
400             //Row 2
401         JLabel tillLabel2 = new JLabel();
402 
403         tillFields[1] = new JTextArea(6,12);
404         tillFields[1].setEditable(false);
405         tillFields[1].getPreferredScrollableViewportSize();
406 
407             //Row 3
408         JLabel tillLabel3 = new JLabel();
409 
410         tillFields[2] = new JTextArea(6,12);
411         tillFields[2].setEditable(false);
412         tillFields[2].getPreferredScrollableViewportSize();
413 
414             //Row 4
415         JLabel tillLabel4 = new JLabel();
```

```
416
417         tillFields[3] = new JTextArea(6,12);
418         tillFields[3].setEditable(false);
419         tillFields[3].getPreferredScrollableViewportSize();
420
421         // Step 2: Set the properties of the components
422         titleLog.setText("Petrol Station Simulation");
423         pumpLabel1.setText("Pump: 1");
424         pumpLabel2.setText("Pump: 2");
425         pumpLabel3.setText("Pump: 3");
426         pumpLabel4.setText("Pump: 4");
427
428         tillLabel1.setText("Till: 1");
429         tillLabel2.setText("Till: 2");
430         tillLabel3.setText("Till: 3");
431         tillLabel4.setText("Till: 4");
432
433         ticksStep.setText("Step:");
434         simMoney.setText("Gained Money: ");
435         simMoney.setHorizontalAlignment(JLabel.CENTER);
436         simLoss.setText("Missed Money: ");
437
438         stopButton.setText("Close Current Simulation");
439
440         // Step 3: Create containers to hold the components
441
442         simulationFrame = new JFrame("Petrol Station Simulation");
443             //Make sure that the red close button doesn't close
444         simulationFrame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
445
```

```
446         // Step 4: Specify LayoutManagers
447             //Simulation Frame
448         simulationFrame.setLayout(new BorderLayout());
449         ((JPanel) simulationFrame.getContentPane()).setBorder(new EmptyBorder(12, 12,
    12, 12));
450 
451             //Title Info Panel
452         JPanel titlePanel = new JPanel();
453         titlePanel.setLayout(new BorderLayout());
454         titlePanel.setBorder(new EmptyBorder(6, 6, 6, 6));
455 
456             //Simulation Info Panel
457         JPanel simPanel = new JPanel();
458         simPanel.setLayout(new BorderLayout());
459         simPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
460 
461             //Main Pump Panel
462         JPanel pumpsPanel = new JPanel();
463         pumpsPanel.setLayout(new BorderLayout());
464         pumpsPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
465 
466             //Stores First 2 Pumps
467         JPanel topPumps = new JPanel();
468         topPumps.setLayout(new BorderLayout());
469         topPumps.setBorder(new EmptyBorder(6, 6, 6, 6));
470             //Stores Last 2 Pumps
471         JPanel bottomPumps = new JPanel();
472         bottomPumps.setLayout(new BorderLayout());
473         bottomPumps.setBorder(new EmptyBorder(6, 6, 6, 6));
474 
```

```
475             //Pump 1
476         JPanel pump1 = new JPanel();
477         pump1.setLayout(new BorderLayout());
478         pump1.setBorder(new EmptyBorder(6, 6, 6, 6));
479             //Left Pump 1
480         JPanel leftSidePump1 = new JPanel();
481         leftSidePump1.setLayout(new BorderLayout());
482         leftSidePump1.setBorder(new EmptyBorder(6, 6, 6, 6));
483             //Right Pump 1
484         JPanel rightSidePump1 = new JPanel();
485         rightSidePump1.setLayout(new BorderLayout());
486         rightSidePump1.setBorder(new EmptyBorder(6, 6, 6, 6));
487 
488             //Pump 2
489         JPanel pump2 = new JPanel();
490         pump2.setLayout(new BorderLayout());
491         pump2.setBorder(new EmptyBorder(6, 6, 6, 6));
492             //Left Pump 2
493         JPanel leftSidePump2 = new JPanel();
494         leftSidePump2.setLayout(new BorderLayout());
495         leftSidePump2.setBorder(new EmptyBorder(6, 6, 6, 6));
496             //Right Pump 2
497         JPanel rightSidePump2 = new JPanel();
498         rightSidePump2.setLayout(new BorderLayout());
499         rightSidePump2.setBorder(new EmptyBorder(6, 6, 6, 6));
500 
501             //Pump 3
502         JPanel pump3 = new JPanel();
503         pump3.setLayout(new BorderLayout());
504         pump3.setBorder(new EmptyBorder(6, 6, 6, 6));
```

```
505             //Left Pump 3
506         JPanel leftSidePump3 = new JPanel();
507         leftSidePump3.setLayout(new BorderLayout());
508         leftSidePump3.setBorder(new EmptyBorder(6, 6, 6, 6));
509             //Right Pump 3
510         JPanel rightSidePump3 = new JPanel();
511         rightSidePump3.setLayout(new BorderLayout());
512         rightSidePump3.setBorder(new EmptyBorder(6, 6, 6, 6));
513 
514             //Pump 4
515         JPanel pump4 = new JPanel();
516         pump4.setLayout(new BorderLayout());
517         pump4.setBorder(new EmptyBorder(6, 6, 6, 6));
518 
519         JPanel leftSidePump4 = new JPanel();
520         leftSidePump4.setLayout(new BorderLayout());
521         leftSidePump4.setBorder(new EmptyBorder(6, 6, 6, 6));
522 
523         JPanel rightSidePump4 = new JPanel();
524         rightSidePump4.setLayout(new BorderLayout());
525         rightSidePump4.setBorder(new EmptyBorder(6, 6, 6, 6));
526 
527             //Main Till Panel
528         JPanel tillsPanel = new JPanel();
529         tillsPanel.setLayout(new BorderLayout());
530         tillsPanel.setBorder(new EmptyBorder(6, 6, 6, 6));
531 
532             //Stores First 2 Tills
533         JPanel topTills = new JPanel();
534         topTills.setLayout(new BorderLayout());
```

```
535         topTills.setBorder(new EmptyBorder(6, 6, 6, 6));
536             //Stores Last 2 Tills
537         JPanel bottomTills = new JPanel();
538         bottomTills.setLayout(new BorderLayout());
539         bottomTills.setBorder(new EmptyBorder(6, 6, 6, 6));
540 
541             //Till 1
542         JPanel till1 = new JPanel();
543         till1.setLayout(new BorderLayout());
544         till1.setBorder(new EmptyBorder(6, 6, 6, 6));
545 
546             //Till 2
547         JPanel till2 = new JPanel();
548         till2.setLayout(new BorderLayout());
549         till2.setBorder(new EmptyBorder(6, 6, 6, 6));
550 
551             //Till 3
552         JPanel till3 = new JPanel();
553         till3.setLayout(new BorderLayout());
554         till3.setBorder(new EmptyBorder(6, 6, 6, 6));
555 
556         //Till 4
557         JPanel till4 = new JPanel();
558         till4.setLayout(new BorderLayout());
559         till4.setBorder(new EmptyBorder(6, 6, 6, 6));
560 
561         // Step 5: Add components to containers
562             //Pumps
563                 //Pump 1
564         leftSidePump1.add(pumpFields[0], BorderLayout.WEST);
```

```
565         leftSidePump1.add(pumpFields[1], BorderLayout.EAST);
566         rightSidePump1.add(pumpFields[2], BorderLayout.WEST);
567         rightSidePump1.add(pumpFields[3], BorderLayout.EAST);
568 
569                 //Merge Left and Right Side
570             pump1.add(pumpLabel1, BorderLayout.NORTH);
571             pump1.add(leftSidePump1, BorderLayout.WEST);
572             pump1.add(rightSidePump1, BorderLayout.EAST);
573 
574                 //Pump 2
575         leftSidePump2.add(pumpFields[4], BorderLayout.WEST);
576         leftSidePump2.add(pumpFields[5], BorderLayout.EAST);
577         rightSidePump2.add(pumpFields[6], BorderLayout.WEST);
578         rightSidePump2.add(pumpFields[7], BorderLayout.EAST);
579                 //Merge Left and Right Side
580             pump2.add(pumpLabel2, BorderLayout.NORTH);
581             pump2.add(leftSidePump2, BorderLayout.WEST);
582             pump2.add(rightSidePump2, BorderLayout.EAST);
583 
584                 //Pump 3
585         leftSidePump3.add(pumpFields[8], BorderLayout.WEST);
586         leftSidePump3.add(pumpFields[9], BorderLayout.EAST);
587         rightSidePump3.add(pumpFields[10], BorderLayout.WEST);
588         rightSidePump3.add(pumpFields[11], BorderLayout.EAST);
589                 //Merge Left and Right Side
590             pump3.add(pumpLabel3, BorderLayout.NORTH);
591             pump3.add(leftSidePump3, BorderLayout.WEST);
592             pump3.add(rightSidePump3, BorderLayout.EAST);
593 
594                 //Pump 4
```

```
595         leftSidePump4.add(pumpFields[12], BorderLayout.WEST);
596         leftSidePump4.add(pumpFields[13], BorderLayout.EAST);
597         rightSidePump4.add(pumpFields[14], BorderLayout.WEST);
598         rightSidePump4.add(pumpFields[15], BorderLayout.EAST);
599                 //Merge Left and Right Side
600             pump4.add(pumpLabel4, BorderLayout.NORTH);
601             pump4.add(leftSidePump4, BorderLayout.WEST);
602             pump4.add(rightSidePump4, BorderLayout.EAST);
603 
604             //Merge Both sides
605         topPumps.add(pump1, BorderLayout.NORTH);
606         topPumps.add(pump2, BorderLayout.SOUTH);
607         bottomPumps.add(pump3, BorderLayout.NORTH);
608         bottomPumps.add(pump4, BorderLayout.SOUTH);
609 
610             //Merge Pumps Together
611         pumpsPanel.add(topPumps, BorderLayout.NORTH);
612         pumpsPanel.add(bottomPumps, BorderLayout.SOUTH);
613 
614         //Tills
615             //Till 1
616             till1.add(tillLabel1, BorderLayout.NORTH);
617             till1.add(tillFields[0], BorderLayout.CENTER);
618 
619             //Till 2
620             till2.add(tillLabel2, BorderLayout.NORTH);
621             till2.add(tillFields[1], BorderLayout.CENTER);
622 
623             //Till 3
624                 //Merge Left and Right Side
```

```
625             till3.add(tillLabel3, BorderLayout.NORTH);
626             till3.add(tillFields[2], BorderLayout.CENTER);
627 
628             //Till 4
629             till4.add(tillLabel4, BorderLayout.NORTH);
630             till4.add(tillFields[3], BorderLayout.WEST);
631 
632             //Merge Both sides
633         topTills.add(till1, BorderLayout.WEST);
634         topTills.add(till2, BorderLayout.EAST);
635         bottomTills.add(till3, BorderLayout.WEST);
636         bottomTills.add(till4, BorderLayout.EAST);
637 
638             //Merge Pumps Together
639         tillsPanel.add(topTills, BorderLayout.NORTH);
640         tillsPanel.add(bottomTills, BorderLayout.SOUTH);
641 
642         //Merge Simulators
643         simPanel.add(pumpsPanel, BorderLayout.WEST);
644         simPanel.add(tillsPanel, BorderLayout.EAST);
645 
646         //Merge Title Panel Componenets
647         titlePanel.add(titleLog, BorderLayout.NORTH);
648         titlePanel.add(ticksStep, BorderLayout.WEST);
649         titlePanel.add(simMoney, BorderLayout.CENTER);
650         titlePanel.add(simLoss, BorderLayout.EAST);
651 
652         //Stuff To Present
653         simulationFrame.add(titlePanel, BorderLayout.NORTH);
654         simulationFrame.add(simPanel, BorderLayout.CENTER);
```

```
655         simulationFrame.add(stopButton, BorderLayout.SOUTH);
656 
657         // Step 6: Arrange to handle events in the user interface
658             //When the red close button is clicked
659         simulationFrame.addWindowListener(new WindowAdapter() {
660             public void windowClosing(WindowEvent e) {
661                 simulationFrame.dispose();
662             }
663         });
664             //When the Stop Button is Pressed
665         stopButton.addActionListener(new ActionListener() {
666             public void actionPerformed(ActionEvent e) {
667                 // Logic to stop
668                 simulationFrame.dispose();
669             }
670         });
671 
672         // Step 7: Display the GUI
673         simulationFrame.pack();
674         simulationFrame.setResizable(false);
675         simulationFrame.setVisible(true);
676 
677         //Call method to grab the parameter values from the Paramter GUI
678         getDetails();
679 
680     }
681 
682     /**
683      * Closes the Program.
684      * <br>This is only used within the parameter GUI.
```

```
685      */
686     private void exitApp() {
687         // Display confirmation dialog before exiting application
688         int response = JOptionPane.showConfirmDialog(mainParameterFrame, "Do you really
    want to quit?", "Quit?",
689                 JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
690 
691         if (response == JOptionPane.YES_OPTION) {
692             System.exit(0);     //Quit
693         }
694     }
695 
696     /**
697      * Gets the details from the Main GUI Parameter Checker
698      */
699     private void getDetails(){
700         // set config values
701         s.config.setScProb((double) pSlider.getValue() / 100);
702         s.config.setMProb((double) pSlider.getValue() / 100);
703         s.config.setFsProb((double) qSlider.getValue() / 100);
704         s.config.setNumPumps((int) Math.pow(2, pumpSlider.getValue()));
705         s.config.setNumTills((int) Math.pow(2, tillSlider.getValue()));
706         s.config.setNumSteps(Integer.parseInt(stepField.getText()));
707         s.config.setPencePerGallon((int) (Double.parseDouble(priceField.getText()) *
    100));
708         s.config.setIsChecked(truckCheck.isSelected());
709 
710         // create PetrolStation
711         s.createPetrolStation();
712 
```

```
713         // run simulation
714         s.simulate(s.config.getNumSteps(),true);
715     }
716 
717     /**
718      * This Displays the info to the Simulation GUI in parts to the JTextFields.
719      * <br>The information is split using a comma delimiter.
720      *
721      * @param info - The Queue Information.
722      *               <br>This is split for the GUI to
723      *               accommodate the JTextFields.
724      */
725     public void display(String info){
726         //Stores the delimited info String to a String Array named 'splitInfo'
727         String[] splitInfo = info.split(",");
728         //Output the info String to the Console
729         System.out.println(info);
730 
731         //Loop through the splitInfo array
732         ticksStep.setText("Step: " + splitInfo[0] + "\t");
733 
734         for (int i = 0; i < splitInfo.length; i++)
735         {
736             //Set the ticksStep JLabel to the current tick
737             ticksStep.setText("Step: " + splitInfo[0] + "\t");
738 
739             //If the number of Pumps are in used based on the Pump Slider value in
    Parameter GUI
740             if(i < ((4 * Math.pow(2, pumpSlider.getValue())))){
741                 //Each pumpField is placed on its respective TextField
```

```
742                 pumpFields[i].setText(splitInfo[i+1]);
743                 pumpFields[i].setText(splitInfo[i+3]);
744             }
745         }
746 
747         //Set all the Fields as Empty
748         tillFields[0].setText("");
749         tillFields[1].setText("");
750         tillFields[2].setText("");
751         tillFields[3].setText("");
752 
753         for (int i = 0; i < splitInfo.length; i++)
754         {
755             if (splitInfo[i].equals("Till1"))
756             {
757                 for (int j = 1; j < splitInfo.length - i; j++)
758                 {
759                     if (!splitInfo[i+j].equals("Till2"))
760                     {
761                         tillFields[0].append(splitInfo[i+j]+"\n");
762                     }
763                     else
764                     {
765                         break;
766                     }
767                 }
768             }
769             if (splitInfo[i].equals("Till2"))
770             {
771                 for (int j = 1; j < splitInfo.length - i; j++)
```

```
772                 {
773                     if (!splitInfo[i+j].equals("Till3"))
774                     {
775                         tillFields[1].append(splitInfo[i+j]+"\n");
776                     }
777                     else
778                     {
779                         break;
780                     }
781                 }
782             }
783             if (splitInfo[i].equals("Till3"))
784             {
785                 for (int j = 1; j < splitInfo.length - i; j++)
786                 {
787                     if (!splitInfo[i+j].equals("Till4"))
788                     {
789                         tillFields[2].append(splitInfo[i+j]+"\n");
790                     }
791                     else
792                     {
793                         break;
794                     }
795                 }
796             }
797             if (splitInfo[i].equals("Till4"))
798             {
799                 for (int j = 1; j < splitInfo.length - i; j++)
800                 {
801                     if (!splitInfo[i+j].equals("Till5"))
```

```
802                     {
803                         tillFields[3].append(splitInfo[i+j]+"\n");
804                     }
805                     else
806                     {
807                         break;
808                     }
809                 }
810             }
811 
812         }
813 
814         simMoney.setText("Gained Money: £" + Double.parseDouble(splitInfo[2])/100);
815         simLoss.setText("Missed Money: £" + Double.parseDouble(splitInfo[1])/100);
816     }
817 
818     /**
819      * Makes the GUI Visible to the user.
820      */
821     public void guiVisible() {
822         mainParameterFrame.setVisible(true);
823     }
824 
825     /**
826      * Initialises all the JTextFields and stores them in an array.
827      */
828     private void initJTxtFld()
829     {
830         //Create JTextField arrays with a text width of 16
831         pumpFields = new JTextField[16];
```

```
832         tillFields = new JTextArea[16];
833         //Loop
834         for (int i = 0; i < 16; i++)
835         {
836             //Create the new JTextField, based on info from arrays
837             pumpFields[i] = new JTextField();
838             if(i >= tillSlider.getValue()){
839                 tillFields[i] = new JTextArea();
840             }
841         }
842     }
843 }
```

```java
package aston.gui;

import aston.simulator.Simulator;

/**
 * The TextView ...
 *
 * @author Tristan P.
 * @author Matas B.
 * @author Kelvin M.
 * @version 03/04/2017
 *
 */
public class TextView {
    /**
     * Text View Constructor<br>
     * This sets the format to print to the information to a file.
     *
     * @param steps how many steps to run for
     * @param p the p value, probabilty of SmallCars and Motorbikes arriving
     * @param q the q value, probabilty of FamilySedans arriving
     * @param pumps the number of pumps there are
     * @param tills the number of tills there are
     * @param gallonPrice the price of fuel in pence/gallon
     * @param trucks whether trucks should be simulated
     */
    public TextView(int steps, double p, double q, int pumps, int tills, int gallonPrice,
boolean trucks)
    {
        //create simulation
```

```
30          Simulator s = new Simulator();
31
32          //set config values
33          s.config.setScProb(p);
34          s.config.setMProb(p);
35          s.config.setFsProb(q);
36          s.config.setNumPumps(pumps);
37          s.config.setNumTills(tills);
38          s.config.setNumSteps(steps);
39          s.config.setPencePerGallon(gallonPrice);
40          s.config.setIsChecked(trucks);
41
42          //create PetrolStation
43          s.createPetrolStation();
44
45          //run simulation
46          s.simulate(s.config.getNumSteps(),false);
47      }
48
49  }
50
```

```java
package aston.resources;
/**
 * This is a class file which contains all the modifiable variables.<br>
 * This also houses the main information and variables of the vehicles and steps.
 *
 * @author Matas B.
 * @author Milton R.
 *
 */
public class Config {
    //Number of steps to run
    public int numSteps = 1;

    //Tick
    public static final int secondsPerTick = 10;

    //SmallCar
    public static final int smallCar_tank = 7;//gallons, tank minimum size
    public static final int smallCar_tankRange = 2;//gallons, tank, size range
    public static final double smallCar_space = 1;//u, size of vehicle in queue
    private double smallCar_probability = 0.03;//probability of vehicle spawning
    public static final int smallCarShoppingTimeLimit = 30;//ticks, max time in queue that allows shopping
    public static final double smallCarShoppingProbability = 0.3;//how likely a car is to shop if allowed
    public static final int smallCarTimeTakenShopping = 12;//ticks, minimum amount of time it will shop for
    public static final int smallCarTimeTakenShoppingRange = 12;//ticks, range of time it wll shop for
    public static final int smallCarMoneySpent = 500; //pence, minimum amount of money
```

```
  spent while shopping
27    public static final int smallCarMoneySpentRange = 500; //pence, range of money spent
  while shopping
28
29    //Motorbike
30    public static final int motorBike_tank = 5;//gallons
31    public static final int motorBike_tankRange = 0;//gallons
32    public static final double motorBike_space = 0.75;//u
33    private double motorBike_probability = 0.03;
34    public static final int motorBikeShoppingTimeLimit = 0;//ticks
35    public static final double motorBikeShoppingProbability = 0;
36    public static final int motorBikeTimeTakenShopping = 0;//ticks
37    public static final int motorBikeTimeTakenShoppingRange = 0;//ticks
38    public static final int motorBikeMoneySpent = 0; //pence
39    public static final int motorBikeMoneySpentRange = 0; //pence
40
41    //FamilySedan
42    public static final int familySedan_tank = 12;//gallons
43    public static final int familySedan_tankRange = 6;//gallons
44    public static final double familySedan_space = 1.5;//u
45    private double familySedan_probability = 0.02;
46    public static final int familySedanShoppingTimeLimit = 60;//ticks
47    public static final double familySedanShoppingProbability = 0.5;
48    public static final int familySedanTimeTakenShopping = 12;//ticks
49    public static final int familySedanTimeTakenShoppingRange = 18;//ticks
50    public static final int familySedanMoneySpent = 800; //pence
51    public static final int familySedanMoneySpentRange = 800; //pence
52
53    //Truck
54    public static final int truck_tank = 30; //gallons, tank minimum size
```

```
55      public static final int truck_tankRange = 10; //gallons, tank, size range
56      public static final int truck_space = 2; //space taken by truck
57      public double truck_probability = 0.02; // truck probability of spawning
58      public static final int truck_shoppingTimeLimit = 48; // truck shopping time
59      public static final double truck_ShoppingProbability = 1;//how likely a truck is to
    shop if allowed
60      public static final int truck_timeTakenShopping = 24; // truck waiting in queues
    range in ticks
61      public static final int truck_timeTakenShoppingRange = 12; // truck shopping range
62      public static final int truck_money_spent = 1500 ; // money spent by truck
63      public static final int truck_money_spent_range = 500; // money range
64      public static final double truck_badservice = 0.2; // bad satisfaction percentage
    drop
65      public static final double truck_goodservice = 0.05; // good satisfaction percentage
    increase
66
67      //Queue
68      public static final int queueSize = 3;
69
70      //Pump
71      public static final int gallonPerTick = 1;//gallon
72      private int numberOfPumps = 2;
73      private int pencePerGallon = 120;//pence
74
75      //truck
76      private boolean isChecked = false;
77
78      //Tills
79      private int numberOftills = 4;
80
```

```java
    //RandomSeed
    public static final int randomSeed = 42;

    /**
     * This allows to get the value correspondent to a Small Car
     * @return The initial probability value of the small car
     */
    public boolean getisChecked()
    {
        return isChecked;
    }

    /**
     * This sets the probability of the Small Cars
     *
     * @param b Double Data Type
     */
    public void setIsChecked(boolean b)
    {
        isChecked = b;
    }

    public double getScProb()
    {
        return smallCar_probability;
    }

    /**
     * This sets the probability of the Small Cars
     *
```

```
111      * @param p Double Data Type
112      */
113     public void setScProb(double p)
114     {
115         smallCar_probability = p;
116     }
117 
118     /**
119      * This allows to get the value correspondent to a Motorbike
120      * @return The initial probability value of the Motobike
121      */
122     public double getMProb()
123     {
124         return motorBike_probability;
125     }
126 
127     /**
128      * This sets the probability of the Motorbike
129      *
130      * @param p Double Data Type
131      */
132     public void setMProb(double p)
133     {
134         motorBike_probability = p;
135     }
136 
137     /**
138      * This allows to get the value correspondent to a Sedan
139      * @return The initial probability value of the Sedan
140      */
```

```
141     public double getFsProb()
142     {
143         return familySedan_probability;
144     }
145 
146     /**
147      * This sets the probability of the Sedan
148      *
149      * @param q Double Data Type
150      */
151     public void setFsProb(double q)
152     {
153         familySedan_probability = q;
154     }
155 
156     /**
157      * This allows to get the value correspondent to a Truck
158      * @return The initial probability value of the Truck
159      */
160     public double getTruckProb() {
161         return truck_probability;
162     }
163     /**
164      * This sets the probability of the Truck
165      *
166      * @param t Double Data Type
167      */
168     public void setTruckprob(double t){
169 
170         truck_probability = t;
```

```
171     }
172
173     /**
174      * This sets the current number of step
175      * @param i Integer Data Type
176      */
177     public void setNumSteps(int i)
178     {
179         numSteps = i;
180     }
181     /**
182      * This allows to get the current value of the current step
183      * @return numSteps: The current Step
184      */
185     public int getNumSteps()
186     {
187         return numSteps;
188     }
189
190
191     /**
192      * Get the current number of pumps, based on the passed parameter from the GUI's
    getDetail method
193      *
194      * @return numberOfPumps
195      */
196     public int getNumPumps()
197     {
198         return numberOfPumps;
199     }
```

```
200
201     /**
202      * Set the current number of pumps, based on the passed parameter from the GUI's
    getDetail method
203      * @param i Integer Data Type
204      */
205     public void setNumPumps(int i)
206     {
207         numberOfPumps = i;
208     }
209
210     /**
211      * Get the price of gallon in pence
212      * @return pencePerGallon
213      */
214     public int getPencePerGallon()
215     {
216         return pencePerGallon;
217     }
218
219     /**
220      * Set the price of gallons in pence
221      * @param i Integer Parameter
222      */
223     public void setPencePerGallon(int i)
224     {
225         pencePerGallon = i;
226     }
227
228     /**
```

```
229     * Get the current number of Tills, based on the passed parameter from the GUI's
    getDetail method
230     * @return numberOfTills
231     */
232    public int getNumTills()
233    {
234        return numberOftills;
235    }
236    /**
237     * Set the current number of pumps, based on the passed parameter from the GUI's
    getDetail method
238     * @param i Integer Value
239     */
240    public void setNumTills(int i)
241    {
242        numberOftills = i;
243    }
244    /**
245     * The Service of the Petrol Station
246     * @param happy Boolean that understands if the customer is satisfied with the
    service.
247     */
248    public void Service(boolean happy)
249    {
250        if(happy)
251        {
252            truck_probability = truck_probability * 1.05;
253        }
254        else
255        {
```

```
256             truck_probability = truck_probability * 0.8;
257         }
258     }
259
260 }
261
```

```java
package aston.resources;
import aston.vehicles.*;
import java.util.LinkedList;
import java.util.Queue;

/**
 * This presents with the Information about the till's queue line length.
 *
 * @author Tristan P.
 * @author Matas B.
 *
 * @version 19/04/2017
 *
 */
public class TillQueue {

    private Queue<Customer> t;
    private int numC = 0;

    /**
     * Till Queue Constructor<br>
     * This creates a new Till in the Shop Class to use.
     */
    public TillQueue()
    {
        t = new LinkedList<Customer>();
    }

    /**
     * Add a customer to the Queue
```

```
31      * @param c Customer
32      */
33     public void addCustomer(Customer c)
34     {
35         t.add(c);
36         numC++;
37     }
38
39     /**
40      * Get the front of the Customer
41      * @return The first person of the Customer being served
42      */
43     public Customer getFrontCustomer()
44     {
45         return t.peek();
46     }
47
48     /**
49      * Remove the front of the customer of the till.
50      */
51     public void removeFrontCustomer()
52     {
53         t.remove();
54         numC--;
55     }
56
57     /**
58      * Accessor Method of the Number of Customers
59      * @return numC Number of Customers
60      */
```

```
61      public int getNumberC()
62      {
63          return numC;
64      }
65
66      /**
67       * This outputs the string information to the console
68       * @return s String concatination
69       */
70      public String toTextString()
71      {
72          String s = "Space Taken: " + numC;
73          for (Customer c : t)
74          {
75              s += ", (" + c.getName() + ")";
76          }
77          return s;
78      }
79      /**
80       * This prints the string information to the GUI
81       * @return s String Concatination
82       */
83      public String toGuiString()
84      {
85          String s = "";
86          for (Customer c : t)
87          {
88              s += c.getName() + ",";
89          }
90
```

```
91          return s;
92      }
93
94
95  }
96
```

```java
package aston.resources;
import java.util.*;
import aston.vehicles.*;

/**
 *
 * This presents with the Information about the Vehicles in line.
 *
 * @author Tristan P.
 * @author Matas B.
 *
 * @version 19/04/2017
 *
 */
public class VehicleQueue {

    private Queue<Vehicle> q;
    private double queueSpace = Config.queueSize;
    public double spaceTaken = 0;
    private int numV = 0;

    /**
     * Constructor<br>
     * This creates a brand new LinkedList Array that stores in Vehicle Objects
     */
    public VehicleQueue()
    {
        q = new LinkedList<Vehicle>();
    }

```

```
31      /**
32       * Adds a vehicle to the queue if there is enough space for it to fit, adds the size
    of the vehicle to the space taken
33       *
34       * @return boolean true if vehicle is added, otherwise false
35       * @param v The vehicle that is going to be added
36       */
37      public boolean addVehicle(Vehicle v)
38      {
39          if (queueSpace >= v.getVehicleSize() + spaceTaken)
40          {
41              spaceTaken += v.getVehicleSize();
42              q.add(v);
43              numV++;
44              v.setVehicleQueue(this);
45              return true;
46          }
47          else
48          {
49              return false;
50          }
51      }
52
53      /**
54       * Gets the vehicle at the front of the queue
55       *
56       * @return Vehicle The vehicle at the front
57       */
58      public Vehicle getFrontVehicle()
59      {
```

```
60          Vehicle v = q.peek();
61          return v;
62      }
63
64      /**
65       * Removes a vehicle from the front of the queue and changes the ammount of space
  taken in the queue
66       */
67      public void removeFrontVehicle()
68      {
69          Vehicle v = q.peek();
70          if (v != null)
71          {
72              q.remove();
73              numV--;
74              spaceTaken -= v.getVehicleSize();
75          }
76      }
77
78      /**
79       * Get the size of the queue
80       *
81       * @return spaceTaken
82       */
83      public double getSize()
84      {
85          return spaceTaken;
86      }
87
88      /**
```

```
 89      * This outputs the string information to the console
 90      * @return s String concatination
 91      */
 92     public String toString()
 93     {
 94         String s = "Space Taken: " + spaceTaken;
 95         for (Vehicle v : q)
 96         {
 97             s += ", (" + v.textToString() + ")";
 98         }
 99         return s;
100     }
101     /**
102      * This prints the string information to the GUI
103      * @return s String Concatination
104      */
105     public String toGuiString()
106     {
107         String s = "";
108         for (Vehicle v : q)
109         {
110             s += v.guiToString() + ",";
111         }
112 
113         //add empty info if less cars
114         for (int i = 4; i > numV; i--)
115         {
116             s += "empty,";
117         }
118         //System.out.println(numV);
```

```
119         return s;
120     }
121
122 }
123
```

```java
package aston.simulator;

import aston.gui.PetrolGUI;
/**
 * This is the starting class of the GUI.
 * <br> This what runs the PetrolGUI class.
 *
 * @author Tristan P.
 * @author Kelvin M.
 * @version 12/04/2017
 *
 */
public class runGUI {

    private static PetrolGUI gui = new PetrolGUI();

    /**
     * Create a simulation and run it for a specified number of steps.
     * This is based on simulator from Lab5
     *
     * @param args The arguements that are taken when run as gui (shouldn't be any)
     */
    public static void main(String[] args) {
        //Make the GUI Visible
        gui.guiVisible();
    }
}

```

```java
package aston.simulator;

import aston.gui.*;

/**
 * Create a simulation and run it for a specified number of steps, based on lab5 simulator
 * takes arguements to change values for how simulator is run.
 *
 * @author Tristan P.
 * @author Matas B.
 *
 * @version 10/04/2017
 */
public class runText {
    /**
     * Main Argument<br>
     * Standard to run a class as an executable
     *
     * @param args the arguements which change how the program will be run in text mode e.g. the p and q values
     */
    public static void main(String[] args) {
        int numSteps = 1;  // By default, run for 1 step

        if (args.length >= 1) {
            numSteps = Integer.parseInt(args[0]);
        }
        if (numSteps <= 0)
        {
```

```
29              numSteps = 1;
30          }
31
32          //change p
33          double pValue = 0.03;
34          if (args.length >= 2)
35          {
36              pValue = Double.parseDouble(args[1]);
37          }
38
39          //change q
40          double qValue = 0.02;
41          if (args.length >= 3)
42          {
43              qValue = Double.parseDouble(args[2]);
44          }
45
46          //change num pumps
47          int numPumps = 2;
48          if (args.length >= 4)
49          {
50              numPumps = Integer.parseInt(args[3]);
51          }
52
53          //change num tills
54          int numTills = 2;
55          if (args.length >= 5)
56          {
57              numTills = Integer.parseInt(args[4]);
58          }
```

```
59
60          //change pence per gallon
61          int gallonPrice = 120;
62          if (args.length >= 6)
63          {
64              gallonPrice = Integer.parseInt(args[5]);
65          }
66
67          boolean trucks = true;
68          if (args.length >= 7)
69          {
70              int i = Integer.parseInt(args[6]);
71              if (i == 1)
72              {
73                  trucks = true;
74              }
75              else
76              {
77                  trucks = false;
78              }
79          }
80
81          TextView textView = new TextView(numSteps, pValue, qValue, numPumps, numTills,
    gallonPrice, trucks);
82      }
83
84  }
85
```

```java
package aston.simulator;
import aston.station.*;
import aston.resources.*;
import aston.gui.*;

/**
 *
 * The Simulator Class is the logic behind the program and details what information to
 the
 * console and GUI.
 *
 * @author Tristan P.
 * @author Matas B.
 *
 * @version 03/04/2017
 *
 */
public class Simulator {

    private PetrolGUI pgui;
    private PetrolStation petrolStation;
    public Config config = new Config();
    private String info = "";
    private int step;

    /**
     * GUI Constructor<br>
     * This is the constructor to create the simulator created
     *
     * @param petrolGUI GUI Simulation Class
```

```
30      */
31     public Simulator(PetrolGUI petrolGUI)
32     {
33         pgui = petrolGUI;
34     }
35
36     /**
37      * Empty Simulator Constructor
38      */
39     public Simulator()
40     {
41         //empty
42     }
43
44     //Methods
45     /**
46      * Simulate the Station in a Tick
47      *
48      * @param numSteps - Current Step
49      * @param gui - Boolean to state if the GUI is present or not
50      */
51     public void simulate(int numSteps, boolean gui)
52     {
53         for(step = 0; step <= numSteps-1; step++) {
54             info = simulateStep(gui);
55             if (gui)
56             {
57                 //Display to the GUI
58                 pgui.display(info);
59             }
```

```
60          }
61      }
62
63      /**
64       * This simulates a step tick and presents the output.
65       *
66       * @version 2.1
67       * @param gui - Boolean to state if the GUI is in use or not
68       * @return info
69       */
70      public String simulateStep(boolean gui)
71      {
72          petrolStation.output.incStep();
73          String info = "";
74          info = toString(gui);
75          if (!gui)
76          {
77              System.out.println(info);
78          }
79          return info;
80
81      }
82
83      /**
84       * This creates an instance of a Petrol Station
85       */
86      public void createPetrolStation()
87      {
88          petrolStation = new PetrolStation(config);
89      }
```

```
 90
 91     /**
 92      * toString Method to give the information out.
 93      *
 94      * @param gui - State if the GUI is true or false
 95      * @return Petrol Station Run Method
 96      */
 97     public String toString(boolean gui){
 98         return petrolStation.run(gui);
 99     }
100 }
101
```

```java
package aston.station;

/**
 * This presents the information with the amount of information required to present to user.
 *
 * @author Tristan P.
 * @version 19/04/2017
 */

public class Output {


    private int numTruck, numSc, numM, numFs, numGallons, totalVSpwnd, currentStep = 0, fuelMoney, lostMoney, additionalMoney;


    /**
     * Set the number of Gallons
     *
     * @param i Integer Data Type
     */
    public void setNumGallons(int i)
    {
        numGallons = i;
    }

    /**
     * Set the Gallon size
     *
```

```
29      * @return numGallons
30      */
31     public int getGallons()
32     {
33         return numGallons;
34     }
35 
36 
37     /**
38      * Get the total Vehicles
39      * @return totalVSpwnd the total amount of Vehicles spawned in the
40      */
41     public int getTotalVehicles()
42     {
43         return totalVSpwnd;
44     }
45 
46     /**
47      * Store Small Car
48      */
49     public void addSC()
50     {
51         numSc++;
52         totalVSpwnd++;
53     }
54 
55     /**
56      * Store number of motorcycles
57      */
58     public void addM()
```

```
59      {
60          numM++;
61          totalVSpwnd++;
62      }
63
64
65      /**
66       * Store family sudans
67       */
68      public void addFS()
69      {
70          numFs++;
71          totalVSpwnd++;
72      }
73
74      /**
75       * Add a Truck to the
76       */
77      public void addTruck()
78      {
79          numTruck++;
80          totalVSpwnd++;
81      }
82
83      /**
84       * Store the Small Car
85       * @return numSc
86       */
87      public int getSC()
88      {
```

```
89          return numSc;
90      }
91
92
93      /**
94       * Get M
95       * @return numM
96       */
97      public int getM()
98      {
99          return numM;
100     }
101
102     /**
103      * Get FS
104      * @return numFS
105      */
106     public int getFS()
107     {
108         return numFs;
109     }
110
111     /**
112      * Get Truck Vehicle Objects
113      * @return numTrucks
114      */
115     public int getTruck()
116     {
117         return numTruck;
118     }
```

```
119
120     /**
121      * Get the current Number of Steps
122      *
123      * @return currentStep Step Counter
124      */
125     public int getNumSteps()
126     {
127         return currentStep;
128     }
129
130     /**
131      * Increment Step Method<br>
132      * This increase the current step counter by 1
133      */
134     public void incStep()
135     {
136         currentStep++;
137     }
138
139     /**
140      * Set the Fuel Money
141      *
142      * @param m Integer Data Type Parameter
143      */
144     public void setFuelMoney(int m)
145     {
146         fuelMoney += m;
147     }
148
```

```
149     /**
150      * Get the Fuel Money
151      * @return fuelMoney
152      */
153     public int getFuelMoney()
154     {
155         return fuelMoney;
156     }
157 
158     /**
159      * Accessor Method of Money Lost during each tick
160      *
161      * @return Money loss
162      */
163     public int getLostMoney() {
164         return lostMoney;
165     }
166 
167     /**
168      * Add the current amount of lost money by the next amount of money lost
169      * @param lostMoney money lost when customer doesnt go to store or vehicle has no
    space at pump and leaves
170      */
171     public void addLostMoney(int lostMoney) {
172         this.lostMoney += lostMoney;
173     }
174 
175     /**
176      * Accessor Method of Addtional Money
177      * @return additionalMoney
```

```
178      */
179     public int getAdditionalMoney() {
180         return additionalMoney;
181     }
182 
183     /**
184      * Mutator Method of Addtional Money
185      * @param additionalMoney the money that is made in the store for a customer
186      */
187     public void addAdditionalMoney(int additionalMoney) {
188         this.additionalMoney += additionalMoney;
189     }
190 
191 }
192 
```

```java
package aston.station;
import java.util.ArrayList;
import aston.vehicles.*;

/**
 * This is the Shop Class which allows to create Shop Simulator
 *
 * @author Tristan P.
 * @author Matas B.
 * @author Kelvin M
 *
 */
public class Shop {

    public Till[] tills;
    private ArrayList<Customer> shoppingCustomers = new ArrayList<Customer>();

    /**
     * Constructor<br>
     * Makes the tills to be used
     * @param numTills Number of tills to use
     */
    public Shop(int numTills)
    {
        tills = new Till[numTills];
        for (int i = 0; i < numTills; i++)
        {
            Till t = new Till(i+1);
            tills[i] = t;
        }
```

```
31      }
32
33      /**
34       * Adds the customer to an array while timer counts down
35       * @param c Current Customer
36       */
37      public void enter(Customer c)
38      {
39          c.setShop(this);
40          shoppingCustomers.add(c);
41      }
42      /**
43       * Customer enters the Till Queue
44       *
45       * @param c Customer
46       */
47      public void tillEnter(Customer c)
48      {
49          c.setShop(this);
50          sendToTill(c);
51      }
52
53      /**
54       * Remove the customer from the array
55       * @param c Customer
56       */
57      public void removeCustomer(Customer c)
58      {
59          shoppingCustomers.remove(c);
60      }
```

```
61      /**
62       * This send the customer to the till
63       *
64       * @param c Current Customer Object
65       */
66      public void sendToTill(Customer c)
67      {
68          int size = tills[0].getQueueSize();
69          Till shortestQueue = tills[0];
70          for (Till t: tills)
71          {
72              if (t.getQueueSize() < size)
73              {
74                  shortestQueue = t;
75                  size = t.getQueueSize();
76              }
77          }
78          shortestQueue.addCustomer(c);
79      }
80
81      /**
82       * This allows the current customer to pass time inside the shop
83       * based on a probability
84       */
85      public void passTime()
86      {
87          for (Till t : tills)
88          {
89              t.serveCustomers();
90          }
```

```
91          if (shoppingCustomers.size() > 0)
92          {
93              ArrayList<Customer> toRemove = new ArrayList<Customer>();
94              for (Customer c : shoppingCustomers)
95              {
96                  if (c.passTime())
97                  {
98                      toRemove.add(c);
99                  }
100             }
101             shoppingCustomers.removeAll(toRemove);
102         }
103
104     }
105
106 }
```

```java
package aston.station;

import java.util.*;
import aston.resources.*;
import aston.vehicles.*;

/**
 * This creates the current utilized GUI that the user will simulate
 *
 * @author Tristan P
 * @author Matas B.
 * @author Jordan L.
 *
 * @version 19/04/2017
 *
 */

public class PetrolStation {

    //Variables
    Config config;
    Vehicle generatedV;

    //instances
    public Random rand = new Random(Config.randomSeed); //temp static
    private Pump[] pumps;
    private Shop shop;
    public Output output = new Output();


```

```
31      /**
32       * Constructor<br>
33       * This creates the inital Petrol Station of the Simulation
34       * @param c Configuration class
35       */
36      public PetrolStation(Config c)
37      {
38          config = c;
39          int numOfPumps = config.getNumPumps();
40          pumps = new Pump[numOfPumps];
41          for (int i = 0; i < numOfPumps; i++)
42          {
43              Pump p = new Pump(i+1);
44              pumps[i] = p;
45          }
46          shop = new Shop(config.getNumTills());
47      }
48
49      /**
50       * Main run method, runs every step/tick.
51       *
52       * @param gui Takes the instance of the gui being used
53       * @return String all the information about the current state of the simulation
54       */
55      public String run(boolean gui)
56      {
57          String information = "";
58          if (!gui)
59          {
60              information += ("\nStep: " + output.getNumSteps() + ", Lost Money: " +
```

```
  output.getLostMoney() + ", Additional Money: " + output.getAdditionalMoney() + ", Fuel
  Money: " + output.getFuelMoney());
61          }
62          else
63          {
64              information += (output.getNumSteps() + "," + output.getLostMoney() + "," +
  (output.getAdditionalMoney()+output.getFuelMoney()) + ",");
65          }
66
67          //customers
68          shop.passTime();
69
70          //make each Pump pump fuel
71          for (Pump p : pumps)
72          {
73              p.pumpFuel();
74          }
75
76          //create a new vehicle
77          if (spawnVehicle())
78          {
79              Vehicle v = generatedV;
80              double size = pumps[0].getQueueSize();
81              Pump shortestQueue = pumps[0];
82              for (Pump p : pumps)
83              {
84                  if (p.getQueueSize() < size)
85                  {
86                      shortestQueue = p;
87                      size = p.getQueueSize();
```

```
88                  }
89              }
90              if (shortestQueue.addVehicleToQueue(v))
91              {
92                  //vehicle goes into a pump queue
93              }
94              else
95              {
96                  System.out.println(v.getName() + " leaves as no space at pump");
97                  output.addLostMoney(v.getTankSize()*config.getPencePerGallon());
98              }
99          }
100
101         //Update output and print Pump info
102         for (Pump p : pumps)
103         {
104             if (!gui)
105             {
106                 information += (p.textToString());
107             }
108             else
109             {
110                 information += (p.guiToString());
111             }
112
113         }
114
115         //Update output and print Till info
116         for (Till t : shop.tills)
117         {
```

```
118             if (!gui)
119             {
120                 information += (t.textToString());
121             }
122             else
123             {
124                 information += (t.getName()+",");
125                 information += (t.guiToString());
126                 if (t.getQueueSize() == 0)
127                 {
128                     information += "empty,";
129                 }
130             }
131 
132         }
133         return information;
134     }
135 
136     /**
137      * Send the customer to the store
138      * @param c Customer Class
139      * @param i Integer is 0 normally, but may be 1 if the customer goes straight to the
    till (motorbikes and unhappy customers)
140      */
141     public void goToShop(Customer c, int i)
142     {
143         if (i == 0)
144         {
145             shop.enter(c);
146         }
```

```
147         else
148         {
149             shop.tillEnter(c);
150             output.addLostMoney(c.getAdditionalMoney());
151         }
152     }
153 
154     /**
155      * Get the current configuration
156      * @return config Config Class
157      */
158     public Config getConfig()
159     {
160         return config;
161     }
162 
163     /**
164      * Creates one random subclass of the vehicle class, based on probabilities in
    config.
165      *
166      * @return true if the number is equivalent to the probablities or false if its
    greater.
167      */
168     private boolean spawnVehicle()
169     {
170         double num = rand.nextDouble();
171         //System.out.println(num);
172 
173         //chose a vehicle
174         if (num < config.getScProb())
```

```
175         {
176             output.addSC();
177             generatedV = new SmallCar(Integer.toString(output.getSC()),this,
    output.getNumSteps());
178             return true;
179         }
180         else if (num < (config.getScProb() + config.getMProb()))
181         {
182             output.addM();
183             generatedV = new Motorbike(Integer.toString(output.getM()),this,
    output.getNumSteps());
184             return true;
185         }
186         else if (num < (config.getScProb() + config.getMProb() + config.getFsProb()))
187         {
188             output.addFS();
189             generatedV = new
    FamilySedan(Integer.toString(output.getFS()),this,output.getNumSteps());
190             return true;
191         }
192         else if (num < (config.getTruckProb() + config.getScProb() + config.getMProb() +
    config.getFsProb()) && (config.getisChecked())) {
193 
194             output.addTruck();
195             generatedV = new Truck(Integer.toString(output.getTruck()),this,
    output.getNumSteps());
196             return true;
197         }
198         else
199         {
```

```
200             return false;
201         }
202     }
203 }
204
```

```java
package aston.station;

import aston.vehicles.*;
import aston.resources.*;

/**
 * Pump Information.<br>
 * This allows to get information about the type of car passing and
 * the
 *
 * @author Kelvin M.
 * @author Tristan P.
 * @author Matas B.
 *
 * @version 19/04/2017
 *
 */
public class Pump {

    int numGallons;
    private Vehicle currentVehicle;
    private VehicleQueue currentQueue = new VehicleQueue();
    int pumpSpeed = Config.gallonPerTick;
    private String name;

    /**
     * Pump Constructor<br>
     * The Pump will be able to
     *
     * @param n - Number of Pumps from the GUI
```

```
31      */
32     public Pump(int n)
33     {
34         name = "Pump"+n;
35     }
36
37     /**
38      * Set the current vehicle to the vehicle in the front of the queue.
39      * <br>Checks if it's null, pumps fuel into the current vehicle
40      */
41     public void pumpFuel()
42     {
43         if (currentQueue.getSize() > 0)
44         {
45             currentVehicle = currentQueue.getFrontVehicle();
46             if(currentVehicle.fillTank(pumpSpeed))
47             {
48                 numGallons ++;
49                 //System.out.println("FILLING TANK");
50             }
51             else
52             {
53                 //removeFrontVehicle();
54                 if (!currentVehicle.hasCustomer())
55                 {
56                     //System.out.println("made customer");
57                     currentVehicle.createCustomer();
58                 }
59                 else
60                 {
```

```
61                      //wait here
62                      //System.out.println("waiting for owner");
63                  }
64              }
65          }
66      }
67
68      /**
69       * Add the Vehicle to the queue
70       *
71       * @param v - The Vehicle Object
72       * @return The selected queue with the vehicle added
73       */
74      public boolean addVehicleToQueue(Vehicle v)
75      {
76          return currentQueue.addVehicle(v);
77      }
78
79      /**
80       * Removes the vehicle at the front of the queue
81       */
82      public void removeFrontVehicle()
83      {
84          currentQueue.removeFrontVehicle();
85      }
86
87      /**
88       * Get the Amount of Gallons in the Pump
89       *
90       * @return The amount of gallons in the pump
```

```
 91      */
 92     public int getNumOfGallons()
 93     {
 94         return numGallons;
 95 
 96     }
 97 
 98     /**
 99      * Print out the Information as a String
100      *
101      * @return A String with the Pump Name and the queue
102      */
103     public String textToString()
104     {
105         return "\n" + name + ":\n" + currentQueue.toString();
106     }
107 
108     /**
109      * Print out the information to the GUI
110      *
111      * @return A method from the queue Object Class
112      */
113     public String guiToString()
114     {
115         return (currentQueue.toGuiString());
116     }
117 
118     /**
119      * Get the current Queue's size
120      *
```

```
121      * @return The current queue's size
122      */
123     public double getQueueSize()
124     {
125         return currentQueue.getSize();
126     }
127 }
128 
```

```java
package aston.station;

import aston.resources.*;
import aston.vehicles.*;
/**
 * This is the Till Class which constructs a Till for the simulatiuon to use
 *
 * @author Matas B.
 * @author Tristan P.
 *
 */
public class Till {

    private TillQueue currentTillQueue = new TillQueue();
    private String name;

    /**
     * Till Constructor<br>
     * This creates a new Till and names the till with a designated number
     *
     * @param n Number
     */
    public Till(int n)
    {
        name = "Till"+n;
    }

    /**
     * Accessor Method of the Name of the Till
     * @return name the name of the till
```

```
31      */
32     public String getName()
33     {
34         return name;
35     }
36 
37     /**
38      * This adds a customer to the current queue of the till
39      * @param c The customer instance that is being added
40      */
41     public void addCustomer(Customer c)
42     {
43         currentTillQueue.addCustomer(c);
44     }
45 
46     /**
47      * This gets the current size of the Queue
48      * @return The current till's queue size
49      */
50     public int getQueueSize()
51     {
52         return currentTillQueue.getNumberC();
53     }
54 
55     /**
56      * Serve the customer's in the till
57      */
58     public void serveCustomers()
59     {
60         if (currentTillQueue.getNumberC() > 0)
```

```
61          {
62              if (currentTillQueue.getFrontCustomer().paid())
63              {
64                  currentTillQueue.getFrontCustomer().leave();
65                  currentTillQueue.removeFrontCustomer();
66              }
67          }
68      }
69      /**
70       * Print the till to Text for TextView Use
71       * @return The name of the till and the Queue
72       */
73      public String textToString()
74      {
75          return "\n" + name + ":\n" + currentTillQueue.toTextString();
76      }
77
78      /**
79       * Print the String to the GUI
80       * @return The Queue of a Till.
81       */
82      public String guiToString()
83      {
84          return (currentTillQueue.toGuiString());
85      }
86  }
87
```

```java
package aston.tests;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Test;
import aston.resources.Config;
import aston.station.PetrolStation;
import aston.vehicles.FamilySedan;



public class FamilySedanTest {

    String n = "2";
    int steps= 30;
    PetrolStation ps = new PetrolStation(new Config());
    FamilySedan testcar = new FamilySedan(n, ps, steps);

    public FamilySedanTest()
    {
        testGetTankSize();
        testGetVehicleSize();
        getShopTimeLimit();
        getShoppingProbability();
        getTimeTakenShopping();
        getMoneySpentShopping();
    }
    @Test
    public void testGetTankSize()
```

```
31      {
32          assertEquals("the tank size must be between 12", 12 ,testcar.getTankSize(), 6);
33      }
34      @Test
35      public void testGetVehicleSize()
36      {
37          assertTrue("Family Sedan's size must be 1.5",( 1.5 == testcar.getVehicleSize()));
38      }
39      @Test
40      public void getShopTimeLimit()
41      {
42          assertEquals("Family Sedan's shop time limit bust be 60", 60,
  ( testcar.getShopTimeLimit()));
43      }
44      @Test
45      public void getShoppingProbability()
46      {
47          assertTrue("Family Sedan's shopping probability must be 0.5 ", (0.5==
  testcar.getShoppingProbability()));
48      }
49      @Test
50      public void getTimeTakenShopping()
51      {
52          assertEquals("Family Sedan's time taken must be between 12 and 24",  12,
  testcar.getTimeTakenShopping(), 12 );
53      }
54      @Test
55      public void getMoneySpentShopping()
56      {
57          assertEquals("Family Sedan's money spent must be betweeen 800 to 800 pence", 800,
```

```
  testcar.getMoneySpentShopping(), 800 );
58     }
59 }
60
```

```java
package aston.tests;

import static org.junit.Assert.assertTrue;

import org.junit.Test;
import aston.resources.Config;
import aston.station.PetrolStation;
import aston.vehicles.Motorbike;


public class MotorbikeTest {

    String n = "1";
    int steps= 30;
    PetrolStation ps = new PetrolStation(new Config());
    Motorbike testcar = new Motorbike(n, ps, steps);

    public MotorbikeTest()
    {
        testGetTankSize();
        testGetVehicleSize();
        getShopTimeLimit();
        getShoppingProbability();
        getTimeTakenShopping();
        getMoneySpentShopping();
    }
    @Test
    public void testGetTankSize()
    {
        assertTrue("Motorbike's the tank size must be 5", (5 == testcar.getTankSize()));
```

```
31      }
32      @Test
33      public void testGetVehicleSize()
34      {
35          assertTrue("Motorbike's size must be 0.75",( 0.75 == testcar.getVehicleSize()));
36      }
37      @Test
38      public void getShopTimeLimit()
39      {
40          assertTrue("Motorbike's shop time limit bust be 0", (0 ==
  testcar.getShopTimeLimit()));
41      }
42      @Test
43      public void getShoppingProbability()
44      {
45          assertTrue("Motorbike's shopping probability must be 0 ", (0 ==
  testcar.getShoppingProbability()));
46      }
47      @Test
48      public void getTimeTakenShopping()
49      {
50          assertTrue("Motorbike's time taken must be 0",  (0 ==
  testcar.getTimeTakenShopping()));
51      }
52      @Test
53      public void getMoneySpentShopping()
54      {
55          assertTrue("Motorbike's money spent must be betweeen 0 pence", (0 ==
  testcar.getMoneySpentShopping()));
56      }
```

```
57
58 }
59
```

```java
package aston.tests;

import static org.junit.Assert.*;
import org.junit.Test;
import aston.station.PetrolStation;
import aston.vehicles.SmallCar;
import aston.vehicles.Vehicle;
import aston.resources.*;

public class SmallCarTest {

    String n = "3";
    int steps= 30;
    PetrolStation ps = new PetrolStation(new Config());
    SmallCar testcar = new SmallCar(n, ps, steps);


    @Test
    public void testGetTankSize()
    {
        assertEquals("cba to type error message", 7 ,testcar.getTankSize(), 2);

    }

    @Test
    public void testGetVehicleSize()
    {
        assertTrue("small car's size must be 1",( 1.0 == testcar.getVehicleSize()));

    }
```

```
31
32
33      //make it loop , and keep using if it's done filling up
34      //when it's false , it is done
35      //store the number of steps taken to fill it up and make sure it lies within the tank
  size range
36
37
38      @Test
39      public void testGetShoppingtime()
40      {
41          assertEquals("small car's shop time limit bust be 30",
  30 ,testcar.getShopTimeLimit(), 12);
42      }
43
44      @Test
45      public void testGetShoppingProbability()
46      {
47          assertTrue("small car's shopping probability must be 0.3 ",
  (0.3==testcar.getShoppingProbability()));
48      }
49
50      @Test
51      public void testGetMoneySpentShopping()
52      {
53          assertEquals("small car's money spent must be betweeen 500 to 1000 pence", 500,
  ((Vehicle) testcar).getMoneySpentShopping(), 500 );
54      }
55
56  }
```

```java
package aston.tests;

import static org.junit.Assert.assertEquals;
import org.junit.Test;
import aston.station.Output;

public class TestOutput {

    Output output = new Output();
    public TestOutput()
    {

        output.addTruck();
        output.addSC();
        output.addFS();
        output.addM();
        output.incStep();
        output.addAdditionalMoney(1);
        output.addLostMoney(1);
        output.setFuelMoney(1);
        output.setNumGallons(1);
        testGetSC();
        testGetM();
        testGetFS();
        testGetTruck();
        testGetNumSteps();
        testGetFuelMoney();
        testGetLostMoney();
        testGetAdditionalMoney();
    }
```

```
31      @Test
32      public void testGetGallons()
33      {
34          assertEquals(" the number of gallons must be the same as 1 ",1,
  output.getGallons());
35      }
36      @Test
37      public void testGetTotalVehicles()
38      {
39          assertEquals(" the number of vehicles must be 1" ,4, output.getTotalVehicles());
40      }
41      @Test
42      public void testGetSC()
43      {
44          assertEquals(" the number of Small Cars must be 1" ,1, output.getSC());
45      }
46      @Test
47      public void testGetM()
48      {
49          assertEquals(" the number of Motorbikes must be 1" ,1, output.getM());
50      }
51      @Test
52      public void testGetFS()
53      {
54          assertEquals(" the number of Family Sedans must be 1", 1 , output.getFS());
55      }
56      @Test
57      public void testGetTruck()
58      {
59          assertEquals(" the number of Trucks must be 1" ,1, output.getTruck());
```

```
60      }
61      @Test
62      public void testGetNumSteps()
63      {
64          assertEquals(" the number of steps must be 1" ,1, output.getNumSteps());
65      }
66      @Test
67      public void testGetFuelMoney()
68      {
69          assertEquals(" the amount of fuel money must be 1" ,1, output.getFuelMoney());
70      }
71      @Test
72      public void testGetLostMoney()
73      {
74          assertEquals(" the amount of lost money must be 1" ,1, output.getLostMoney());
75      }
76      @Test
77      public void testGetAdditionalMoney() {
78          assertEquals(" the amount of vehicles must be 1" ,1,
    output.getAdditionalMoney());
79      }
80
81  }
82
```

```java
package aston.tests;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

import aston.resources.Config;
import aston.station.PetrolStation;
import aston.vehicles.Truck;

public class TruckTestClass {

    String n = "4";
    int steps= 30;
    PetrolStation ps = new PetrolStation(new Config());
    Truck testcar = new Truck(n, ps, steps);

    @Test
    public void testGetTankSize()
    {
        assertEquals("truck's tank size is 30", 30,testcar.getTankSize(), 10);

    }

    @Test
    public void testGetVehicleSize()
    {
        assertTrue("Truck's size must be 2",( 2 == testcar.getVehicleSize()));

```

```
31      }
32
33
34      //make it loop , and keep using if it's done filling up
35      //when it's false , it is done
36      //store the number of steps taken to fill it up and make sure it lies within the tank
  size range
37
38
39      @Test
40      public void testGetShoppingtime()
41      {
42          assertEquals("Truck's shop time limit must be 48",
  48 ,testcar.getShopTimeLimit(), 12);
43      }
44
45      @Test
46      public void testGetShoppingProbability()
47      {
48          assertTrue("Trucks's shopping probability must be 0.3 ", (1 ==
  testcar.getShoppingProbability()));
49      }
50
51      @Test
52      public void testGetMoneySpentShopping()
53      {
54          assertEquals("Trucks's money spent must be betweeen 500 to 1000 pence", 1500,
  testcar.getMoneySpentShopping(), 500 );
55      }
56
```

```
57 }
58
```

```java
package aston.vehicles;

import aston.station.*;
/**
 * This is the Customer Class. This creates a customer based on the Vehicle
 *
 * @author Tristan P.
 * @author Matas B.
 */
public class Customer {

    private Vehicle ownedVehicle;
    private String name;
    private int shoppingTime;
    private Shop shop;
    private int additionalMoney;
    private int tillTime = 13;  //min time in till queue + 1


    /**
     * Constructor<br>
     * Create a new customer with a set time to shop and state that the customer owns this vehicle
     *
     * @param ov Instance of the vehicle which the customer owns
     * @param shoppingTime how long the customer will spend shopping
     * @param happy boolean whether the customer is happy
     * @param shopMoney how much the customer will spend in the shop
     */
    public Customer(Vehicle ov, int shoppingTime, boolean happy, int shopMoney)
```

```
30      {
31          setAdditionalMoney(shopMoney);
32          ownedVehicle = ov;
33          this.shoppingTime = shoppingTime;
34          String vName = ov.getName();
35          name = "Customer" + vName.substring(0,1) + vName.replaceAll("\\D+","");
36          if(vName.equals("Truck"))
37          {
38              ov.petrolStation.getConfig().Service(happy);
39          }
40      }
41
42      /**
43       * Get the amount of time shopping
44       *
45       * @return shoppingTime
46       */
47      public int getShoppingTime()
48      {
49          return shoppingTime;
50      }
51
52      /**
53       * Set the Shop
54       *
55       * @param s Shop Class
56       */
57      public void setShop(Shop s)
58      {
59          shop = s;
```

```
60      }
61
62      /**
63       * This states if the customer is withint he amount of time shopping
64       *
65       * @return true if the Customer is going to the till
66       * @return false if it is over or below the tick counter
67       */
68      public boolean passTime()
69      {
70          if (shoppingTime <= 0)
71          {
72              tillTime += ownedVehicle.petrolStation.rand.nextInt(6);
73              shop.sendToTill(this);
74              System.out.println(name + " goes to till");
75              return true;
76          }
77          else
78          {
79              shoppingTime--;
80              return false;
81          }
82      }
83
84      /**
85       * Paid method where the customer is still in the Shop or the queue.
86       *
87       * @return true Customer  has no more time or has paid in the Till
88       * @return false Customer still has time to shop around or are in the queue.
89       */
```

```
 90     public boolean paid()
 91     {
 92         tillTime--;
 93         if (tillTime > 0)
 94         {
 95             return false;
 96         }
 97         else
 98         {
 99             return true;
100         }
101     }
102 
103     /**
104      * Leave the Station
105      */
106     public void leave()
107     {
108         PetrolStation ps = ownedVehicle.petrolStation;
109         ps.output.addAdditionalMoney(additionalMoney);
110 
    ps.output.setFuelMoney(ownedVehicle.getTankSize()*ps.getConfig().getPencePerGallon());
111         ownedVehicle.vLeave();
112         System.out.println(name + " drives away");
113     }
114     /**
115      * Grab the name of the customer
116      * @return name Name of the customer
117      */
118     public String getName()
```

```
119     {
120         return name;
121     }
122 
123     /**
124      * Accessor Method for Additional Money
125      * @return additionalMoney
126      */
127     public int getAdditionalMoney() {
128         return additionalMoney;
129     }
130 
131     /**
132      * Mutator Method for additional Money
133      * @param additionalMoney sets the value of the money that will be spent by the
    customer in the shop
134      */
135     public void setAdditionalMoney(int additionalMoney) {
136         this.additionalMoney = additionalMoney;
137     }
138 }
139 
```

```java
package aston.vehicles;

import aston.resources.*;
import aston.station.*;
/**
 * The Family Sedan is a subclass of the Abstract Vehicle Class.<br>
 * This will cointain information that comes from the Config File.
 *  *
 * @author Tristan P.
 * @author Kelvin M.
 * @author Matas B.
 *
 */
public class FamilySedan extends Vehicle{
    /**
     * Family Sedan Constructor that is calling the super-class: Vehicle
     *
     * @param n the number of trucks taht have been created, used to generate unique name
     * @param ps the instance of the petrol station running
     * @param steps What step the vehicle was created on, used to work out how long spent in queue
     */
    public FamilySedan(String n, PetrolStation ps, int steps){
        super(ps,steps);

        tank = Config.familySedan_tank;

        space = Config.familySedan_space;
        shopTimeLimit = Config.familySedanShoppingTimeLimit;
        shoppingProbability = Config.familySedanShoppingProbability;
```

```
30          timeTakenShopping = Config.familySedanTimeTakenShopping +
  petrolStation.rand.nextInt(Config.familySedanTimeTakenShoppingRange);
31          moneySpentShopping = Config.familySedanMoneySpent +
  petrolStation.rand.nextInt(Config.familySedanMoneySpentRange);
32          currentTank = 0;
33          name = "FamilySedan" + n;
34      }
35  }
36
```

```java
package aston.vehicles;

import aston.resources.Config;
import aston.station.PetrolStation;

/**
 * The Motorbike is a subclass of the Abstract Vehicle Class.<br>
 * This will cointain information that comes from the Config File.
 *
 * @author Matas B.
 * @author Tristan P.
 *
 */
public class Motorbike extends Vehicle{

    /**
     * Motorbike Constructor that is calling the super-class: Vehicle
     *
     * @param n the number of trucks taht have been created, used to generate unique name
     * @param ps the instance of the petrol station running
     * @param steps What step the vehicle was created on, used to work out how long spent in queue
     */
    public Motorbike(String n, PetrolStation ps, int steps){
        super(ps, steps);
        tank = Config.motorBike_tank;

        space = Config.motorBike_space;
        shopTimeLimit = Config.motorBikeShoppingTimeLimit;
        shoppingProbability = Config.motorBikeShoppingProbability;
```

```
30
31          timeTakenShopping = Config.motorBikeTimeTakenShopping;
32
33          moneySpentShopping = Config.motorBikeMoneySpent;
34
35          currentTank = 0;
36          name = "Motorbike" + n;
37      }
38  }
```

```java
package aston.vehicles;

import aston.resources.Config;
import aston.station.PetrolStation;
/**
 * The Small Car is a subclass of the Abstract Vehicle Class.<br>
 * This will cointain information that comes from the Config File.
 *  *
 * @author Tristan P.
 * @author Matas B.
 *
 */
public class SmallCar extends Vehicle{

    /**
     * SmallCar Constructor that is calling the super-class: Vehicle
     *
     * @param n String
     * @param ps PetrolStation Class
     * @param steps Integer
     */
    public SmallCar(String n, PetrolStation ps, int steps){
        super(ps,steps);
        if (Config.smallCar_tankRange > 0)
        {
            tank = Config.smallCar_tank +
petrolStation.rand.nextInt(Config.smallCar_tankRange);
        }

        space = Config.smallCar_space;
```

```
30          shopTimeLimit = Config.smallCarShoppingTimeLimit;
31          shoppingProbability = Config.smallCarShoppingProbability;
32          timeTakenShopping = Config.smallCarTimeTakenShopping +
  petrolStation.rand.nextInt(Config.smallCarTimeTakenShoppingRange);
33          moneySpentShopping = Config.smallCarMoneySpent +
  petrolStation.rand.nextInt(Config.smallCarMoneySpentRange);
34          name = "SmallCar" + n;
35      }
36
37  }
38
```

```java
package aston.vehicles;

import aston.resources.Config;
import aston.station.PetrolStation;

/**
 * Truck Class<br>
 * This is a Level 1 feature that is controlled by the Config Class.
 *
 * @author Milton R.
 * @author Matas B.
 *
 */
public class Truck extends Vehicle{
    protected int waitingTickLimit;
    protected double arrivalProbability;

    /**
     * Motorbike Constructor that is calling the super-class: Vehicle
     *
     * @param n the number of trucks taht have been created, used to generate unique name
     * @param ps the instance of the petrol station running
     * @param steps What step the vehicle was created on, used to work out how long spent in queue
     */
    public Truck(String n, PetrolStation ps, int steps) {
        super(ps, steps);
        waitingTickLimit = Config.truck_shoppingTimeLimit;
        tank = Config.truck_tank + petrolStation.rand.nextInt(Config.truck_tankRange);
        space = Config.truck_space;
```

```
30          shopTimeLimit = Config.truck_shoppingTimeLimit;
31          shoppingProbability = Config.truck_ShoppingProbability;
32          timeTakenShopping = Config.truck_timeTakenShopping +
  petrolStation.rand.nextInt(Config.truck_timeTakenShoppingRange);
33          moneySpentShopping = Config.truck_money_spent +
  petrolStation.rand.nextInt(Config.truck_money_spent_range);
34          currentTank = 0;
35          name = "Truck" + n;
36      }
37
38  }
39
```

```java
package aston.vehicles;

import aston.station.*;
import aston.resources.*;
/**
 * A superclass for vehicles, extended by Family Sedan, Motorbike, Small Car.
 * The vehicle takes up a certain amount of space and it has a tank that needs to be filled.
 *
 * @author Kelvin M.
 * @author Tristan P.
 * @author Matas B.
 * @author Milton R.
 *
 * @version 1.7 03/05/2017
 *
 */
public abstract class Vehicle{
    //Variables
    protected int steps;
    protected String name;
    protected int tank;                         //Vehicle's Tank Size
    protected double space;                     //How much space the vehicle uses in queue
    protected int currentTank;                  //Vehicle's Current Tank
    //protected int arrivalProbability;         //Probability that the vehicle will arrive
    protected int shopTimeLimit;                //Time that which pump must take less than in order for the customer to shop
    protected double shoppingProbability;       //Probability that the customer will
```

```
  shop if time is under shopTimeLimit
27    protected int timeTakenShopping;             //Time that customer will spend shopping
28    //protected int timeTakenShoppingRange;      //Range of time in shop
29    protected int moneySpentShopping;            //Minimum Value in Payment
30    //protected int moneySpentShoppingRange;     //Min + Random Value in Payment
31    //protected Random random;                   //Random Class
32    protected Customer customer;
33    protected boolean hasCustomer = false;
34    protected VehicleQueue vQ;
35
36    public PetrolStation petrolStation;
37
38    /**
39     * Constructor of the Vehicle
40     *
41     * @param ps Petrol Station
42     * @param steps What step the vehicle was created on, used to work out how long
  spent in queue
43     */
44    public Vehicle(PetrolStation ps, int steps){
45        petrolStation = ps;
46        tank = 0;
47        space = 0;
48        this.steps = steps;
49        currentTank = 0;
50        shopTimeLimit = 2;
51        timeTakenShopping = 1;
52        moneySpentShopping = 2;
53    }
54
```

```
55      /**
56       * Fill the Vehicle's Tank
57       *
58       * @param pumpSpd the speed at which fuel is pumped into the vehicle
59       * @return boolean whether it pumped fuel or not
60       */
61      public boolean fillTank(int pumpSpd)
62      {
63          if (currentTank < tank)
64          {
65              currentTank += pumpSpd;
66              //System.out.println("tank: "+currentTank);
67              return true;
68          }
69          else
70          {
71              //System.out.println("tfull");
72              return false;
73          }
74      }
75
76      /**
77       * Gets the money that has been spent by the customers belonging to the Vehicle.
78       *
79       * @return Money spent by the customers.
80       */
81      public int getMoneySpentShopping()
82      {
83          return moneySpentShopping;
84
```

```
85      }
86
87      /**
88       * Gets the time taken to shop, by the customers belonging to the Vehicle.
89       * @return The time takan to shop.
90       */
91      public int getTimeTakenShopping()
92      {
93          return timeTakenShopping;
94      }
95
96      /**
97       * Gets the probability that a belonging to the Vehicle will shop.
98       * @return The time taken to shop.
99       */
100     public double getShoppingProbability()
101     {
102         return shoppingProbability;
103     }
104
105
106     /**
107      * Gets the maximum amount of time that a customer belonging to the vehicle will
    spend shopping.
108      * @return The the time limit.
109      */
110     public int getShopTimeLimit()
111     {
112         return shopTimeLimit;
113     }
```

```
114     public int getTankSize()
115     {
116         return tank;
117     }
118 
119 
120 
121 
122     /**
123      * Gets the current amount of fuel in the vehicles tank.
124      * @return The current amount of fuel.
125      */
126     public int getCurrentTank()
127     {
128         return currentTank;
129 
130     }
131 
132     /**
133      * Gets the size of the vehicle in terms of the queue.
134      * @return The space taken.
135      */
136     public double getVehicleSize()
137     {
138         return space;
139     }
140 
141     /**
142      * Create a Customer
143      */
```

```
144     public void createCustomer()
145     {
146         hasCustomer = true;
147         int shoppingTime = 0;
148         if (name.contains("SmallCar"))
149         {
150             shoppingTime = Config.smallCarTimeTakenShopping
151                     + petrolStation.rand.nextInt(Config.smallCarTimeTakenShoppingRange);
152         }
153         else if (name.contains("Motorbike"))
154         {
155                 shoppingTime = Config.motorBikeTimeTakenShopping;
156         }
157         else if (name.contains("FamilySedan"))
158         {
159             shoppingTime = Config.familySedanTimeTakenShopping
160                     +
    petrolStation.rand.nextInt(Config.familySedanTimeTakenShoppingRange);
161         }
162         else if (name.contains("Truck")) // creates customer truck
163         {
164             shoppingTime = Config.truck_timeTakenShopping
165                     + petrolStation.rand.nextInt(Config.truck_timeTakenShoppingRange);
166         }
167         customer = new Customer(this, shoppingTime, getHappy(), moneySpentShopping);
168         if (!(petrolStation.rand.nextDouble() <= shoppingProbability))
169         {
170             petrolStation.goToShop(customer,0);
171             System.out.println(customer.getName() + " goes into the store");
172         }
```

```
173         else
174         {
175             petrolStation.goToShop(customer,1);
176             System.out.println(customer.getName() + " goes straight to the till");
177         }
178     }
179 
180     /**
181      * Get how happy a customer/vehicle is
182      * @return true Number of steps is within the shop time limit
183      * @return false Number of steps outside the shop time limit
184      */
185     private boolean getHappy()
186     {
187         int currentStep = petrolStation.output.getNumSteps();
188         currentStep -= steps;
189         if (currentStep <= shopTimeLimit)
190         {
191             return true;
192         }
193         else return false;
194     }
195 
196     /**
197      * Accessor for the Vehicle Name
198      * @return name
199      */
200     public String getName()
201     {
202         return name;
```

```
203     }
204
205     /**
206      * Boolean if a Vehicle has a Customer out of the Vehicle
207      * @return hasCustomer
208      */
209     public boolean hasCustomer()
210     {
211         return hasCustomer;
212     }
213
214     /**
215      * Set the Queue of Vehicles.
216      * @param vq VehicleQueue Class
217      */
218     public void setVehicleQueue(VehicleQueue vq)
219     {
220         vQ = vq;
221     }
222     /**
223      * Vehicle Leaves the Queue of Vehicles
224      */
225     public void vLeave()
226     {
227         vQ.removeFrontVehicle();
228     }
229
230     /**
231      * Gets the toString representation of the class.
232      * @return The string representation.
```

```
233      */
234     public String toString()
235     {
236         return "tank: "+tank+", space: "+space+", shop time limit: "
237     + shopTimeLimit + ", shopping probability: "
238                 + shoppingProbability + ", time taken shopping: "
239     + timeTakenShopping + ", moneySpentShopping: " + moneySpentShopping ;
240     }
241 
242     /**
243      * Returns info for text view
244      *
245      * @return String Representation for Console
246      */
247     public String textToString()
248     {
249         return name + ", Tank: " + currentTank + "/" + tank + ", Size: " + space;
250     }
251 
252     /**
253      * Returns info for GUI view
254      *
255      * @return String Representation for GUI
256      */
257     public String guiToString()
258     {
259         return name + ": " + currentTank + "/" + tank;
260     }
261 
262 }
```

263