```
 1:
 2: package cakesolutioncorrectversion;
 3:
 4: import java.util.ArrayList;
 5: import java.util.List;
 6:
 7: /**
 8:  *
 9:  *
10:  * Class Baklavaci extends a superclass SuperLocation which that class
11:  * implements the interface Location. This class represents one of the locations
12:  * of the game. it initializes questionList of its own, and it has a description
13:  * of which world the player is at.
14:  *
15:  * @author Cakesolutiongroup
16:  * @version 1.0
17:  * @since ( 01 may 2013)
18:  */
19: public class Baklavaci extends SuperLocation {
20:
21:     /**
22:      * getDescription method that returns the name of the world the player is
23:      * at.
24:      *
25:      * @return String "Baklavaci" - the name of the world where the player is
26:      * at.
27:      */
28:     @Override
29:     public String getDescription() {
30:         return "Baklavaci";
31:     }
32:
33:     /**
34:      * Method initializeBQ that generates questions and choices for the world,
35:      * Baklavaci.
36:      * @return Null
37:      * @param Null
38:      *
39:      */
40:     public void initializeQuestionList() {
41:
42:         Question q1 = new Question("On which continent does Turkey lie?");
43:
44:         Choice bQ1C1 = new Choice();
45:         bQ1C1.choice = "Asia";
46:         bQ1C1.isCorrectChoice = false;
47:
```

```
48:         Choice bQ1C2 = new Choice();
49:         bQ1C2.choice = "Europe";
50:         bQ1C2.isCorrectChoice = false;
51:
52:         Choice bQ1C3 = new Choice();
53:         bQ1C3.choice = "Europe and Asia";
54:         bQ1C3.isCorrectChoice = true;
55:
56:         Choice bQ1C4 = new Choice();
57:         bQ1C4.choice = "Middle East";
58:         bQ1C4.isCorrectChoice = false;
59:
60:         questionList.add(q1);
61:         q1.choices.add(bQ1C1);
62:         q1.choices.add(bQ1C2);
63:         q1.choices.add(bQ1C3);
64:         q1.choices.add(bQ1C4);
65:
66:         Question q2 = new Question("What is the name of the Turkish capital?");
67:
68:         Choice bQ2C1 = new Choice();
69:         bQ2C1.choice = "Istanbul";
70:         bQ2C1.isCorrectChoice = false;
71:
72:         Choice bQ2C2 = new Choice();
73:         bQ2C2.choice = "Ankara";
74:         bQ2C2.isCorrectChoice = true;
75:
76:         Choice bQ2C3 = new Choice();
77:         bQ2C3.choice = "Marmaris";
78:         bQ2C3.isCorrectChoice = false;
79:
80:         Choice bQ2C4 = new Choice();
81:         bQ2C4.choice = "Izmir";
82:         bQ2C4.isCorrectChoice = false;
83:
84:         questionList.add(q2);
85:         q2.choices.add(bQ2C1);
86:         q2.choices.add(bQ2C2);
87:         q2.choices.add(bQ2C3);
88:         q2.choices.add(bQ2C4);
89:
90:         Question q3 = new Question("What is Hamam?");
91:
92:         Choice bQ3C1 = new Choice();
93:         bQ3C1.choice = "Turkish bath";
94:         bQ3C1.isCorrectChoice = true;
```

```
 95:
 96:         Choice bQ3C2 = new Choice();
 97:         bQ3C2.choice = "A dessert";
 98:         bQ3C2.isCorrectChoice = false;
 99:
100:         Choice bQ3C3 = new Choice();
101:         bQ3C3.choice = "A sport discipline";
102:         bQ3C3.isCorrectChoice = false;
103:
104:         Choice bQ3C4 = new Choice();
105:         bQ3C4.choice = "A casserole dish";
106:         bQ3C4.isCorrectChoice = false;
107:
108:         questionList.add(q3);
109:         q3.choices.add(bQ3C1);
110:         q3.choices.add(bQ3C2);
111:         q3.choices.add(bQ3C3);
112:         q3.choices.add(bQ3C4);
113:
114:         Question q4 = new Question("What is baklava?");
115:
116:         Choice bQ4C1 = new Choice();
117:         bQ4C1.choice = "A sweet dessert";
118:         bQ4C1.isCorrectChoice = true;
119:
120:         Choice bQ4C2 = new Choice();
121:         bQ4C2.choice = "A strong and bitter coffee";
122:         bQ4C2.isCorrectChoice = false;
123:
124:         Choice bQ4C3 = new Choice();
125:         bQ4C3.choice = "A spicy alcoholic drink";
126:         bQ4C3.isCorrectChoice = false;
127:
128:         Choice bQ4C4 = new Choice();
129:         bQ4C4.choice = "A tangy breakfast treat";
130:         bQ4C4.isCorrectChoice = false;
131:
132:         questionList.add(q4);
133:         q4.choices.add(bQ4C1);
134:         q4.choices.add(bQ4C2);
135:         q4.choices.add(bQ4C3);
136:         q4.choices.add(bQ4C4);
137:
138:         Question q5 = new Question("What is the national sport of Turkey?");
139:
140:         Choice bQ5C1 = new Choice();
141:         bQ5C1.choice = "Football";
```

```
142:         bQ5C1.isCorrectChoice = false;
143:
144:         Choice bQ5C2 = new Choice();
145:         bQ5C2.choice = "Oil wrestling";
146:         bQ5C2.isCorrectChoice = true;
147:
148:         Choice bQ5C3 = new Choice();
149:         bQ5C3.choice = "Weightlifting";
150:         bQ5C3.isCorrectChoice = false;
151:
152:         Choice bQ5C4 = new Choice();
153:         bQ5C4.choice = "Football";
154:         bQ5C4.isCorrectChoice = false;
155:
156:         questionList.add(q5);
157:         q5.choices.add(bQ5C1);
158:         q5.choices.add(bQ5C2);
159:         q5.choices.add(bQ5C3);
160:         q5.choices.add(bQ5C4);
161:
162:         Question q6 = new Question("When was the Republic of Turkey founded?");
163:
164:         Choice bQ6C1 = new Choice();
165:         bQ6C1.choice = "1823";
166:         bQ6C1.isCorrectChoice = false;
167:
168:         Choice bQ6C2 = new Choice();
169:         bQ6C2.choice = "1868";
170:         bQ6C2.isCorrectChoice = false;
171:
172:         Choice bQ6C3 = new Choice();
173:         bQ6C3.choice = "1923";
174:         bQ6C3.isCorrectChoice = true;
175:
176:         Choice bQ6C4 = new Choice();
177:         bQ6C4.choice = "1945";
178:         bQ6C4.isCorrectChoice = false;
179:
180:         questionList.add(q6);
181:         q6.choices.add(bQ6C1);
182:         q6.choices.add(bQ6C2);
183:         q6.choices.add(bQ6C3);
184:         q6.choices.add(bQ6C4);
185:
186:     }
187: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: /**
 4:  *
 5:  * This class just holds the sugarLevel.
 6:  *
 7:  * @author CakeSolutionGroup
 8:  * @version 1.0
 9:  * @since (01 may 2013)
10:  *
11:  */
12: public class Cake {
13:
14:     int sugarLevel = 10;
15: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: import cakesolutioncorrectversion.Location;
 4: import java.util.ArrayList;
 5: import java.util.List;
 6:
 7: /**
 8:  * This class generates an arrayList called myLocations which adds all existing
 9:  * locations in the game. i.e : Baklavaci The class also implements the
10:  * interface World.
11:  *
12:  * @author CakeSolutionGroup
13:  * @since (01 may 2013)
14:  */
15: public class Cakistan implements World {
16:
17:     List<Location> myLocations = new ArrayList<Location>();
18:
19:     /**
20:      * This creates a list of all the locations and connects to World
21:      *
22:      */
23:     public Cakistan() {
24:         Location baklavaci = new Baklavaci();
25:         Location lagkagehuset = new Lagkagehuset();
26:         Location lePetiteEclaire = new LePetiteEclaire();
27:         myLocations.add(baklavaci);
28:         myLocations.add(lagkagehuset);
29:         myLocations.add(lePetiteEclaire);
30:
31:         baklavaci.setNeighbor(lagkagehuset);
32:         lagkagehuset.setNeighbor(lePetiteEclaire);
33:         lePetiteEclaire.setNeighbor(baklavaci);
34:
35:     }
36:
37:     /**
38:      * Method getLocations that returns arrayList myLocations.
39:      *
40:      * @return myLocations of type arrayList
41:      */
42:     @Override
43:     public List<Location> getLocations() {
44:         return myLocations;
45:     }
46:
47:     /**
```

```
48:      * This method has a for loop that goes through all 3 locations and
49:      * determines if the QuestionList in each location is empty or not, meaning
50:      * that the questions are all asked or not.
51:      *
52:      * @return true/ false
53:      */
54:     public boolean isOutOfQuestions() {
55:
56:         for (int i = 0; i < myLocations.size(); i++) {
57:
58:             boolean isEmpty = myLocations.get(i).getQuestionList().isEmpty();
59:             if (!isEmpty) {
60:                 return false;
61:             }
62:         }
63:         return true;
64:     }
65: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: /**
 4:  * This class has String choice, and a boolean isCorrectChoice, which is used in
 5:  * initializeBQ, initializeQuestions, initializeLPEQ.
 6:  *
 7:  * @author CakeSolutionGroup
 8:  * @version 1.0
 9:  * @since (01 may 2013)
10:  */
11: public class Choice {
12:
13:     String choice;
14:     boolean isCorrectChoice;
15: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: import java.util.ArrayList;
 4: import java.util.InputMismatchException;
 5: import java.util.List;
 6: import java.util.Scanner;
 7:
 8: /**
 9:  * This class is the Controller part of MVC (Model View Controller). It controls
10:  * the actions in the game. Class Controller declares "world" of type Cakistan,
11:  * "player1" of type Sugarman, "the view" of type View and "r" of type Random.
12:  *
13:  * @author CakeSolutionGroup
14:  * @version 1.0
15:  * @since (01 may 2013)
16:  */
17: public class Controller {
18:
19:     World world;
20:     Sugarman player1;
21:     View theView;
22:
23:     /**
24:      * Method main instantiate game of type Controller. declares method
25:      * startGame and runGame from game.
26:      *
27:      * @param args
28:      */
29:     public static void main(String[] args) {
30:
31:         Controller game = new Controller();
32:         game.startGame();
33:         game.runGame();
34:
35:     }
36:
37:     /**
38:      * Method startGame instantiate world, player1 and the view. It gets the
39:      * location from class Cakistan, and set the starting point from the first
40:      * element in Location arrayList, which is Baklavaci.
41:      */
42:     public void startGame() {
43:
44:
45:         // Creates worlds
46:         world = new Cakistan();
47:
```

```
48:         // Creates player
49:         player1 = new Sugarman();
50:
51:         theView = new View();
52:
53:
54:         //get starting location from world
55:         List<Location> locations = world.getLocations();
56:         Location start = locations.get(0);
57:
58:         player1.setLocation(start);
59:
60:     }
61:
62:     /**
63:      * This method randomizes questions for each location and if the answer to
64:      * the question was right award the user with a cake, which is equal to 10
65:      * points.
66:      *
67:      * @param null
68:      *
69:      */
70:     public void askRandomQuestion() {
71:
72:         Question q = player1.getLocation().getRandomQuestion();
73:         theView.printOutQuestion(q);
74:         theView.printEnterAnswer();
75:         int answer = userInput(q.choices.size());
76:         boolean wasCorrectAnswer = player1.isAnswerCorrect(q, answer);
77:
78:         if (wasCorrectAnswer) {
79:             theView.printYouGotACake();
80:         } else {
81:             theView.printNoCake();
82:         }
83:
84:     }
85:
86:     /**
87:      * This method is a boolean which returns true, if either Sugarman has lost,
88:      * or won or all locations are out of questions[all questions are asked].
89:      *
90:      * @return true
91:      */
92:     private boolean isGameOver() {
93:
94:         return player1.hasSugarmanLost() || player1.hasSugarmanWon() || world.isOutOfQuestions();
```

```
 95:
 96:     }
 97:
 98:     /**
 99:      * a method of type boolean which returns false if user's input is less than
100:      * 1 or it's greater than max. otherwise returns true.
101:      *
102:      * @param input
103:      * @param max
104:      * @return true/ false
105:      */
106:     public boolean isInputValid(int input, int max) {
107:
108:         if (input < 1 || input > max) {
109:             return false;
110:         }
111:         return true;
112:     }
113:
114:     /**
115:      * This method just calls the view for printing messages in case of:
116:      * Sugarman lost, Sugarman won or locations are out of questions.
117:      *
118:      * @param null
119:      * @return null
120:      */
121:     public void printGameOver() {
122:
123:         if (player1.hasSugarmanLost()) {
124:             theView.printSugarmanLost();
125:         } else if (player1.hasSugarmanWon()) {
126:             theView.printSugarmanWon();
127:         } else {
128:             theView.printRanOutOfQuestions();
129:         }
130:
131:     }
132:
133:     /**
134:      * Method runGame run the game for as long as the player hasn't won or hasn't
135:      * lost or hasn't run out of questions in the game or hasn't exited the game.
136:      *
137:      *
138:      */
139:     public void runGame() {
140:         while (!isGameOver()) {
141:             theView.playerStatus(player1);
```

```
142:             theView.printToMoveOrToStay(player1.getLocation().getDescription(),
143:                     player1.getLocation().getNeighbor().getDescription());
144:             int decision = userInput(3);
145:             if (decision == 1) { //user chose to stay and answer questions
146:
147:                 ArrayList<Question> currentQuestionList = player1.getLocation().getQuestionList();
148:                 if (currentQuestionList.isEmpty()) {
149:                     theView.printNoQuestionsLeft();
150:                     continue;
151:                 }
152:
153:                 askRandomQuestion();
154:
155:             } else if (decision == 2) { //user chose to move to next location
156:                 player1.move(player1.getLocation().getNeighbor());
157:             }
158:             else if (decision == 3){ //user chose to exit the game
159:                 theView.printExitGame();
160:                 System.exit(1);
161:             }
162:         } //end of while loop
163:         printGameOver();
164:     }
165:
166:     /**
167:      * Scans the user's input and checks if it's valid or not.
168:      *
169:      * @param max of type integer
170:      * @return user's input of type integer
171:      */
172:     public int userInput(int max) {
173:
174:
175:         while (true) {
176:
177:             int input = 0;
178:             Scanner userInput = new Scanner(System.in);
179:
180:             try {
181:
182:                 input = userInput.nextInt();
183:             } catch (InputMismatchException ex) {
184:                 theView.printRetypeInput(max);
185:                 continue;
186:             }
187:
188:             if (isInputValid(input, max)) {
```

```
189:
190:                 return input;
191:
192:             } else {
193:                 theView.printRetypeInput(max);
194:             }
195:         }
196:
197:
198:     }
199: }
```

```
 1:
 2: package cakesolutioncorrectversion;
 3:
 4: /**
 5:  *Class GameObject extends the interface Localizable.
 6:  * The class used to hold a boolean variable declared as 'canBeTaken'.
 7:  * This class was created for the initial PacMan layer, where the idea was that
 8:  * the player was able to go around and take cakes. So canBeTaken would have
 9:  * returned true, once the player have answered the question correctly.
10:  *
11:  * @author CakeSolutionGroup
12:  * @version 1.0
13:  * @since (01 may 2013)
14:  *
15:  */
16: public interface GameObject extends Localizable {
17:
18:
19:
20: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: import java.util.ArrayList;
 4: import java.util.List;
 5:
 6: /**
 7:  * Class Lagkagehuset extends a superclass SuperLocation, which implements the
 8:  * interface Location. Lagkagehuset initializes a questionList and has a
 9:  * description of which location the player is at.
10:  *
11:  * @author CakeSolutionGroup
12:  * @version 1.0
13:  * @since (01 may 2013)
14:  */
15: public class Lagkagehuset extends SuperLocation {
16:
17:     /**
18:      * This is the description of the location Lagkagehuset.
19:      *
20:      * @param Null
21:      * @return String Lagkagehuset.
22:      */
23:     @Override
24:     public String getDescription() {
25:         return "Lagkagehuset";
26:     }
27:
28:     /**
29:      * Method initializeQuestions that generates questions and choices for the
30:      * world, Lagkagehuset.
31:      *
32:      * @return Null
33:      * @param Null
34:      *
35:      */
36:     public void initializeQuestionList() {
37:
38:         Question q1 = new Question("What is the name of the current Queen of Denmark?");
39:
40:         Choice lkhQ1C1 = new Choice();
41:         lkhQ1C1.choice = "Margrethe";
42:         lkhQ1C1.isCorrectChoice = true;
43:
44:         Choice lkhQ1C2 = new Choice();
45:         lkhQ1C2.choice = "Ingrid";
46:         lkhQ1C2.isCorrectChoice = false;
47:
```

```
48:         Choice lkhQ1C3 = new Choice();
49:         lkhQ1C3.choice = "Beatrice";
50:         lkhQ1C3.isCorrectChoice = false;
51:
52:         Choice lkhQ1C4 = new Choice();
53:         lkhQ1C4.choice = "Mary";
54:         lkhQ1C4.isCorrectChoice = false;
55:
56:         questionList.add(q1);
57:         q1.choices.add(lkhQ1C1);
58:         q1.choices.add(lkhQ1C2);
59:         q1.choices.add(lkhQ1C3);
60:         q1.choices.add(lkhQ1C4);
61:
62:
63:         Question q2 = new Question("What is the national dish of Denmark?");
64:
65:         Choice lkhQ2C1 = new Choice();
66:         lkhQ2C1.choice = "A Danish";
67:         lkhQ2C1.isCorrectChoice = false;
68:
69:         Choice lkhQ2C2 = new Choice();
70:         lkhQ2C2.choice = "Shooting Star";
71:         lkhQ2C2.isCorrectChoice = false;
72:
73:         Choice lkhQ2C3 = new Choice();
74:         lkhQ2C3.choice = "Meatballs";
75:         lkhQ2C3.isCorrectChoice = true;
76:
77:         Choice lkhQ2C4 = new Choice();
78:         lkhQ2C4.choice = "Open-faced sandwiches";
79:         lkhQ2C4.isCorrectChoice = false;
80:
81:         questionList.add(q2);
82:         q2.choices.add(lkhQ2C1);
83:         q2.choices.add(lkhQ2C2);
84:         q2.choices.add(lkhQ2C3);
85:         q2.choices.add(lkhQ2C4);
86:
87:         Question q3 = new Question("Which island is the biggest in Denmark?");
88:
89:         Choice lkhQ3C1 = new Choice();
90:         lkhQ3C1.choice = "Greenland";
91:         lkhQ3C1.isCorrectChoice = false;
92:
93:         Choice lkhQ3C2 = new Choice();
94:         lkhQ3C2.choice = "Zealand";
```

```
95:         lkhQ3C2.isCorrectChoice = true;
96:
97:         Choice lkhQ3C3 = new Choice();
98:         lkhQ3C3.choice = "Funen";
99:         lkhQ3C3.isCorrectChoice = false;
100:
101:        Choice lkhQ3C4 = new Choice();
102:        lkhQ3C4.choice = "Vendsyssel-Thy";
103:        lkhQ3C4.isCorrectChoice = false;
104:
105:        questionList.add(q3);
106:        q3.choices.add(lkhQ3C1);
107:        q3.choices.add(lkhQ3C2);
108:        q3.choices.add(lkhQ3C3);
109:        q3.choices.add(lkhQ3C4);
110:
111:        Question q4 = new Question("Which bird is the Danish national bird?");
112:
113:        Choice lkhQ4C1 = new Choice();
114:        lkhQ4C1.choice = "The alaudidae lark";
115:        lkhQ4C1.isCorrectChoice = false;
116:
117:        Choice lkhQ4C2 = new Choice();
118:        lkhQ4C2.choice = "the mute swan";
119:        lkhQ4C2.isCorrectChoice = true;
120:
121:        Choice lkhQ4C3 = new Choice();
122:        lkhQ4C3.choice = "the shiny blackbird";
123:        lkhQ4C3.isCorrectChoice = false;
124:
125:        Choice lkhQ4C4 = new Choice();
126:        lkhQ4C4.choice = "the red swallow";
127:        lkhQ4C4.isCorrectChoice = false;
128:
129:        questionList.add(q4);
130:        q4.choices.add(lkhQ4C1);
131:        q4.choices.add(lkhQ4C2);
132:        q4.choices.add(lkhQ4C3);
133:        q4.choices.add(lkhQ4C4);
134:
135:        Question q5 = new Question("When was Denmark liberated after WW2?");
136:
137:        Choice lkhQ5C1 = new Choice();
138:        lkhQ5C1.choice = "may 4th 1945";
139:        lkhQ5C1.isCorrectChoice = false;
140:
141:        Choice lkhQ5C2 = new Choice();
```

```
142:         lkhQ5C2.choice = "july 18th 1945";
143:         lkhQ5C2.isCorrectChoice = false;
144:
145:         Choice lkhQ5C3 = new Choice();
146:         lkhQ5C3.choice = "june 7th 1945";
147:         lkhQ5C3.isCorrectChoice = false;
148:
149:         Choice lkhQ5C4 = new Choice();
150:         lkhQ5C4.choice = "may 5th 1945";
151:         lkhQ5C4.isCorrectChoice = true;
152:
153:         questionList.add(q5);
154:         q5.choices.add(lkhQ5C1);
155:         q5.choices.add(lkhQ5C2);
156:         q5.choices.add(lkhQ5C3);
157:         q5.choices.add(lkhQ5C4);
158:
159:         Question q6 = new Question("When did Copenhagen become the capital of Denmark?");
160:
161:         Choice lkhQ6C1 = new Choice();
162:         lkhQ6C1.choice = "Mid 14th century";
163:         lkhQ6C1.isCorrectChoice = false;
164:
165:         Choice lkhQ6C2 = new Choice();
166:         lkhQ6C2.choice = "Mid 15th century";
167:         lkhQ6C2.isCorrectChoice = true;
168:
169:         Choice lkhQ6C3 = new Choice();
170:         lkhQ6C3.choice = "Mid 16th century";
171:         lkhQ6C3.isCorrectChoice = false;
172:
173:         Choice lkhQ6C4 = new Choice();
174:         lkhQ6C4.choice = "Mid 17th century";
175:         lkhQ6C4.isCorrectChoice = false;
176:
177:         questionList.add(q6);
178:         q6.choices.add(lkhQ6C1);
179:         q6.choices.add(lkhQ6C2);
180:         q6.choices.add(lkhQ6C3);
181:         q6.choices.add(lkhQ6C4);
182:
183:         Question q7 = new Question("Why did hamlet say that \"something was rotten in the State of Denmark\"? ");
184:
185:         Choice lkhQ7C1 = new Choice();
186:         lkhQ7C1.choice = "He disliked the smell of fish";
187:         lkhQ7C1.isCorrectChoice = false;
188:
```

```
189:         Choice lkhQ7C2 = new Choice();
190:         lkhQ7C2.choice = "All was not well at the top of the political hierarchy";
191:         lkhQ7C2.isCorrectChoice = true;
192:
193:         Choice lkhQ7C3 = new Choice();
194:         lkhQ7C3.choice = "He wanted a warmer climate";
195:         lkhQ7C3.isCorrectChoice = false;
196:
197:         Choice lkhQ7C4 = new Choice();
198:         lkhQ7C4.choice = "He was no fan of bacon";
199:         lkhQ7C4.isCorrectChoice = false;
200:
201:         questionList.add(q7);
202:         q7.choices.add(lkhQ7C1);
203:         q7.choices.add(lkhQ7C2);
204:         q7.choices.add(lkhQ7C3);
205:         q7.choices.add(lkhQ7C4);
206:
207:         Question q8 = new Question("When was H. C. Andersen born?");
208:
209:         Choice lkhQ8C1 = new Choice();
210:         lkhQ8C1.choice = "1792";
211:         lkhQ8C1.isCorrectChoice = false;
212:
213:         Choice lkhQ8C2 = new Choice();
214:         lkhQ8C2.choice = "1802";
215:         lkhQ8C2.isCorrectChoice = true;
216:
217:         Choice lkhQ8C3 = new Choice();
218:         lkhQ8C3.choice = "1812";
219:         lkhQ8C3.isCorrectChoice = false;
220:
221:         Choice lkhQ8C4 = new Choice();
222:         lkhQ8C4.choice = "1822";
223:         lkhQ8C4.isCorrectChoice = false;
224:
225:         questionList.add(q8);
226:         q8.choices.add(lkhQ8C1);
227:         q8.choices.add(lkhQ8C2);
228:         q8.choices.add(lkhQ8C3);
229:         q8.choices.add(lkhQ8C4);
230:
231:         Question q9 = new Question("Which programming language was invented by the Dane Bjarne Stroustrup?");
232:
233:         Choice lkhQ9C1 = new Choice();
234:         lkhQ9C1.choice = "PHP";
235:         lkhQ9C1.isCorrectChoice = false;
```

```
236:
237:         Choice lkhQ9C2 = new Choice();
238:         lkhQ9C2.choice = "C++";
239:         lkhQ9C2.isCorrectChoice = true;
240:
241:         Choice lkhQ9C3 = new Choice();
242:         lkhQ9C3.choice = "C#";
243:         lkhQ9C3.isCorrectChoice = false;
244:
245:         Choice lkhQ9C4 = new Choice();
246:         lkhQ9C4.choice = "Perl";
247:         lkhQ9C4.isCorrectChoice = false;
248:
249:         questionList.add(q9);
250:         q9.choices.add(lkhQ9C1);
251:         q9.choices.add(lkhQ9C2);
252:         q9.choices.add(lkhQ9C3);
253:         q9.choices.add(lkhQ9C4);
254:
255:     }
256: }
```

```
    1: package cakesolutioncorrectversion;
    2:
    3: import java.util.List;
    4: import java.util.ArrayList;
    5:
    6: /**
    7:  * Class LePetiteEclaire extends a superclass SuperLocation which that class
    8:  * implements the interface Location. This class initializes questionList of its
    9:  * own, and it has a description of which world the player is at.
   10:  *
   11:  * @author CakeSolutionGroup
   12:  * @version 1.0
   13:  * @since (01 may 2013)
   14:  */
   15: public class LePetiteEclaire extends SuperLocation {
   16:
   17:     /**
   18:      * This is the description of the location LePetiteEclaire
   19:      *
   20:      * @return String Le petite Eclaire.
   21:      */
   22:     @Override
   23:     public String getDescription() {
   24:         return "Le Petite Eclaire";
   25:     }
   26:
   27:     /**
   28:      * Method initializeLPEQ that generates questions and choices for the world,
   29:      * LePetiteEclaire.
   30:      *@return Null
   31:      * @param Null
   32:      */
   33:     public void initializeQuestionList() {
   34:
   35:         Question q1 = new Question("What is the name of the current French republic?");
   36:
   37:         Choice lpeQ1C1 = new Choice();
   38:         lpeQ1C1.choice = "The Third Republic";
   39:         lpeQ1C1.isCorrectChoice = false;
   40:
   41:         Choice lpeQ1C2 = new Choice();
   42:         lpeQ1C2.choice = "The Fourth Republic";
   43:         lpeQ1C2.isCorrectChoice = false;
   44:
   45:         Choice lpeQ1C3 = new Choice();
   46:         lpeQ1C3.choice = "The Fifth Republic";
   47:         lpeQ1C3.isCorrectChoice = true;
```

```
48:
49:         Choice lpeQ1C4 = new Choice();
50:         lpeQ1C4.choice = "The Sixth Republic";
51:         lpeQ1C4.isCorrectChoice = false;
52:
53:         questionList.add(q1);
54:         q1.choices.add(lpeQ1C1);
55:         q1.choices.add(lpeQ1C2);
56:         q1.choices.add(lpeQ1C3);
57:         q1.choices.add(lpeQ1C4);
58:
59:
60:         Question q2 = new Question("What is the Bastille Day?");
61:
62:         Choice lpeQ2C1 = new Choice();
63:         lpeQ2C1.choice = "The National Day";
64:         lpeQ2C1.isCorrectChoice = true;
65:
66:         Choice lpeQ2C2 = new Choice();
67:         lpeQ2C2.choice = "An extended shopping day";
68:         lpeQ2C2.isCorrectChoice = false;
69:
70:         Choice lpeQ2C3 = new Choice();
71:         lpeQ2C3.choice = "The liberation of France in WW2";
72:         lpeQ2C3.isCorrectChoice = false;
73:
74:         Choice lpeQ2C4 = new Choice();
75:         lpeQ2C4.choice = "The occupation of France in WW2";
76:         lpeQ2C4.isCorrectChoice = false;
77:
78:         questionList.add(q2);
79:         q2.choices.add(lpeQ2C1);
80:         q2.choices.add(lpeQ2C2);
81:         q2.choices.add(lpeQ2C3);
82:         q2.choices.add(lpeQ2C4);
83:
84:
85:         Question q3 = new Question("How many regional languages is there in France?");
86:
87:         Choice lpeQ3C1 = new Choice();
88:         lpeQ3C1.choice = "2";
89:         lpeQ3C1.isCorrectChoice = false;
90:
91:         Choice lpeQ3C2 = new Choice();
92:         lpeQ3C2.choice = "5";
93:         lpeQ3C2.isCorrectChoice = false;
94:
```

```
 95:         Choice lpeQ3C3 = new Choice();
 96:         lpeQ3C3.choice = "7";
 97:         lpeQ3C3.isCorrectChoice = true;
 98:
 99:         Choice lpeQ3C4 = new Choice();
100:         lpeQ3C4.choice = "9";
101:         lpeQ3C4.isCorrectChoice = false;
102:
103:         questionList.add(q3);
104:         q3.choices.add(lpeQ3C1);
105:         q3.choices.add(lpeQ3C2);
106:         q3.choices.add(lpeQ3C3);
107:         q3.choices.add(lpeQ3C4);
108:
109:         Question q4 = new Question("The population of France is roughly: ");
110:
111:         Choice lpeQ4C1 = new Choice();
112:         lpeQ4C1.choice = "52 million";
113:         lpeQ4C1.isCorrectChoice = false;
114:
115:         Choice lpeQ4C2 = new Choice();
116:         lpeQ4C2.choice = "65 million";
117:         lpeQ4C2.isCorrectChoice = true;
118:
119:         Choice lpeQ4C3 = new Choice();
120:         lpeQ4C3.choice = "78 million";
121:         lpeQ4C3.isCorrectChoice = false;
122:
123:         Choice lpeQ4C4 = new Choice();
124:         lpeQ4C4.choice = "100 million";
125:         lpeQ4C4.isCorrectChoice = false;
126:
127:         questionList.add(q4);
128:         q4.choices.add(lpeQ4C1);
129:         q4.choices.add(lpeQ4C2);
130:         q4.choices.add(lpeQ4C3);
131:         q4.choices.add(lpeQ4C4);
132:
133:         Question q5 = new Question("France is: ");
134:
135:         Choice lpeQ5C1 = new Choice();
136:         lpeQ5C1.choice = "The largest country in the EU";
137:         lpeQ5C1.isCorrectChoice = true;
138:
139:         Choice lpeQ5C2 = new Choice();
140:         lpeQ5C2.choice = "Larger than Italy but smaller than Spain";
141:         lpeQ5C2.isCorrectChoice = false;
```

```
142:
143:         Choice lpeQ5C3 = new Choice();
144:         lpeQ5C3.choice = "Twice as large as Germany";
145:         lpeQ5C3.isCorrectChoice = false;
146:
147:         Choice lpeQ5C4 = new Choice();
148:         lpeQ5C4.choice = "As large as the state of Nevada";
149:         lpeQ5C4.isCorrectChoice = false;
150:
151:         questionList.add(q5);
152:         q5.choices.add(lpeQ5C1);
153:         q5.choices.add(lpeQ5C2);
154:         q5.choices.add(lpeQ5C3);
155:         q5.choices.add(lpeQ5C4);
156:
157:         Question q6 = new Question("Which three are the largest French cities?");
158:
159:         Choice lpeQ6C1 = new Choice();
160:         lpeQ6C1.choice = "Paris, Nice, Lyon";
161:         lpeQ6C1.isCorrectChoice = false;
162:
163:         Choice lpeQ6C2 = new Choice();
164:         lpeQ6C2.choice = "Paris, Marseille, Lille";
165:         lpeQ6C2.isCorrectChoice = false;
166:
167:         Choice lpeQ6C3 = new Choice();
168:         lpeQ6C3.choice = "Paris, Lyon, Marseille";
169:         lpeQ6C3.isCorrectChoice = true;
170:
171:         Choice lpeQ6C4 = new Choice();
172:         lpeQ6C4.choice = "Paris, Toulouse, Lille";
173:         lpeQ6C4.isCorrectChoice = false;
174:
175:         questionList.add(q6);
176:         q6.choices.add(lpeQ6C1);
177:         q6.choices.add(lpeQ6C2);
178:         q6.choices.add(lpeQ6C3);
179:         q6.choices.add(lpeQ6C4);
180:
181:
182:
183:     }
184: }
```

```
 1: package cakesolutioncorrectversion;
 2:
 3: import cakesolutioncorrectversion.Location;
 4:
 5: /**
 6:  * Localizable interface holds getLocation and setLocation method of type Location.
 7:  *
 8:  * @author CakeSolutionGroup
 9:  * @version 1.0
10:  * @since (01 may 2013)
11:  */
12: public interface Localizable {
13:
14:     /**
15:      * Returns the current location
16:      */
17:     Location getLocation();
18:
19:     /**
20:      * Sets the location
21:      */
22:     boolean setLocation(Location location);
23: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: import java.util.ArrayList;
 8: import java.util.List;
 9:
10: /**
11:  * Location interface sets contracts for SuperLocation which eventually sets contracts for each 3 locations.
12:  * @author Cakesolutiongroup
13:  * @version 1.0
14:  * @since ( 01 may 2013)
15:  */
16: public interface Location {
17:
18:     String getDescription();
19:
20:     public Location getNeighbor();
21:
22:     public ArrayList<Question> getQuestionList();
23:
24:     public Question getRandomQuestion();
25:
26:     public void setNeighbor(Location l);
27:
28:
29:
30:
31: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: import cakesolutioncorrectversion.Location;
 8:
 9: /**
10:  *
11:  * @author CakeSolutionGroup
12:  * @version 1.0
13:  * @since (01 may 2013)
14:  */
15: public interface PlayerController extends Localizable {
16:
17:     /**
18:      * Changes the Sugar level of the player as specified by the difference
19:      *
20:      * @param difference change in Sugar level.
21:      */
22:     void changeSugarLevel(int difference);
23:
24:     /**
25:      * Player can not eat the cake
26:      *
27:      * @param object of type Cake
28:      * @return true if the question is answered correctly, false otherwise
29:      */
30:     public void dontEatCake(Cake cake);
31:
32:     /**
33:      * Player can eat the cake
34:      *
35:      * @param object of type Cake
36:      * @return true if the question is answered correctly, false otherwise
37:      */
38:     public void eatCake(Cake cake);
39:
40:     /**
41:      * Returns the sugar level of the player
42:      *
43:      * @return
44:      */
45:     int getSugarlevel();
46:
47:     /**
```

```
48:      * Moves the player around to a new location.
49:      *
50:      * @param direction direction to be moved to
51:      * @Return true if new location can be sat as requested, false otherwise
52:      */
53:     public void move(Location location);
54: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: /**
 8:  * @author CakeSolutionGroup
 9:  * @version 1.0
10:  * @since (01 may 2013)
11:  */
12: public class Players {
13:     //Delete this! But to exam remember superclasses if we wanted more players beside sugarman.
14:
15: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: import java.util.ArrayList;
 8:
 9: /**
10:  * the class 'Question' contains a constructor that holds the questionText for
11:  * the specific questions. The class also contains the variables needed when
12:  * answering questions.
13:  *
14:  * @author CakeSolutionGroup
15:  * @version 1.0 (2 May 2013)
16:  */
17: public class Question {
18:
19:     String question;
20:     ArrayList<Choice> choices = new ArrayList<Choice>();
21:     Cake cake = new Cake();
22:
23:     /**
24:      * the constructor creates a new instance of question
25:      *
26:      * @param questionText of type String
27:      *
28:      */
29:     public Question(String questionText) {
30:         this.question = questionText;
31:
32:     }
33: }
```

```
    1: package cakesolutioncorrectversion;
    2:
    3: import cakesolutioncorrectversion.Players;
    4: import cakesolutioncorrectversion.PlayerController;
    5: import cakesolutioncorrectversion.Location;
    6:
    7: /**
    8:  * Sugarman is a class that extends Players and implements PlayerController. The
    9:  * class contains the methods needed for the player to play the game, such as
   10:  * keeping score (the sugar level), eating or not eating the cakes and the
   11:  * location of the player.
   12:  *
   13:  * @author CakeSolutionGroup
   14:  * @version 1.0 (2 May 2013)
   15:  */
   16: public class Sugarman extends Players implements PlayerController {
   17:
   18:     private Location sugarmanLocation;
   19:     private int currentSugarLevel = 50;
   20:
   21:     /**
   22:      * changeSugarLevel changes the currentSugarLevel
   23:      *
   24:      * @param difference, which is an integer variable
   25:      * @return Null
   26:      */
   27:     @Override
   28:     public void changeSugarLevel(int difference) {
   29:
   30:         currentSugarLevel = currentSugarLevel + difference;
   31:     }
   32:
   33:     /**
   34:      * dontEatCake sets the value for the integer 'difference' in the method
   35:      * changeSugarLevel, whenever the player answers a question incorrectly
   36:      *
   37:      * @param cake of class type Cake
   38:      * @return Null
   39:      */
   40:     @Override
   41:     public void dontEatCake(Cake cake) {
   42:         changeSugarLevel(-cake.sugarLevel);
   43:
   44:     }
   45:
   46:     /**
   47:      * eatCake sets the value for the integer 'difference' in the method
```

```
48:      * changeSugarLevel, whenever the player answers a question correctly
49:      *
50:      * @param cake of class type Cake
51:      * @return Null
52:      */
53:     public void eatCake(Cake cake) {
54:         changeSugarLevel(cake.sugarLevel);
55:     }
56:
57:     /**
58:      * getLocation returns the current location of sugarman
59:      *
60:      * @param Null
61:      * @returns sugarmanLocation of class type Location
62:      */
63:     @Override
64:     public Location getLocation() {
65:         return sugarmanLocation;
66:
67:     }
68:
69:     /**
70:      * setLocation returns true whenever the location equals sugarmanLocation
71:      *
72:      * @param location of class type Location
73:      * @return boolean
74:      */
75:     @Override
76:     public boolean setLocation(Location location) {
77:
78:         sugarmanLocation = location;
79:         return true;
80:
81:     }
82:
83:     /**
84:      * getSugarLevel returns the currentSugarLevel, which keeps track of the
85:      * player's progress
86:      *
87:      * @param null
88:      * @returns the currentSugarLevel of type integer
89:      */
90:     @Override
91:     public int getSugarlevel() {
92:
93:         return currentSugarLevel;
94:     }
```

```
 95:
 96:     /**
 97:      * if the currentSugarLevel of sugarman is equal or above 100, it returns
 98:      * hasWon.
 99:      *
100:      * @return hasWon
101:      */
102:     public boolean hasSugarmanWon() {
103:         boolean hasWon = currentSugarLevel >= 100;
104:         return hasWon;
105:
106:     }
107:
108:     /**
109:      * if the currentSugarLevel of sugarman is equal or less than 0, it returns
110:      * hasLost.
111:      *
112:      * @return hasLost
113:      */
114:     public boolean hasSugarmanLost() {
115:         boolean hasLost = currentSugarLevel <= 0;
116:         return hasLost;
117:     }
118:
119:     /**
120:      * isAnswerCorrect is a boolean method which returns true if the user's
121:      * answer is correct. and it returns false otherwise.
122:      *
123:      * @param q
124:      * @param answer
125:      * @return true/ false
126:      */
127:     public boolean isAnswerCorrect(Question q, int answer) {
128:
129:         if (q.choices.get(answer - 1).isCorrectChoice) {
130:             eatCake(q.cake);
131:             return true;
132:         } else {
133:             dontEatCake(q.cake);
134:             return false;
135:         }
136:     }
137:
138:     /**
139:      * move is a method that determines whether the player can move to the
140:      * desired location.
141:      *
```

```
142:      * @param direction
143:      * @return boolean
144:      */
145:     @Override
146:     public void move(Location location) {
147:         sugarmanLocation = location;
148:
149:     }
150: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: //import apple.laf.JRSUIConstants;
 8: import cakesolutioncorrectversion.Location;
 9: import java.util.ArrayList;
10: import java.util.List;
11: import java.util.Random;
12:
13: /**
14:  * The class 'SuperLocation' is a super class, which implements Location. It's
15:  * purpose is to create the methods for placing the locations in relation to
16:  * each other. It also contains methods that hold information about the
17:  * different locations.
18:  *
19:  * @author CakeSolutionGroup
20:  * @version 1.0 (1 May 2013)
21:  */
22: abstract class SuperLocation implements Location {
23:
24:     private Location neighbor;
25:     Random r = new Random();
26:     ArrayList<Question> questionList = new ArrayList<Question>();
27:
28:     /**
29:      * Class constructor that initializes the questions for the location.
30:      */
31:     public SuperLocation() {
32:
33:         initializeQuestionList();
34:
35:     }
36:
37:
38:
39:     /**
40:      * getQuestionList method that returns question list for the location.
41:      *
42:      * @return questionList
43:      */
44:     @Override
45:     public ArrayList<Question> getQuestionList() {
46:
47:         return questionList;
```

```
48:     }
49:
50:     @Override
51:     public Question getRandomQuestion() {
52:
53:         int randomNumber = r.nextInt(questionList.size());
54:         Question q = questionList.get(randomNumber);
55:         questionList.remove(randomNumber);
56:
57:         return q;
58:     }
59:
60:
61:       abstract void initializeQuestionList();
62:
63:
64:
65:     /**
66:      *
67:      * @param l
68:      * @return neighbor of type location
69:      */
70:     @Override
71:     public void setNeighbor(Location l) {
72:         neighbor = l;
73:     }
74:
75:     /**
76:      * @param null
77:      * @return null
78:      */
79:     @Override
80:     public Location getNeighbor() {
81:         return neighbor;
82:
83:     }
84: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: /**
 8:  * The Class 'View' is the view in the MVC model. The class prints information
 9:  * about the status of the game, and the tasks at hand during gameplay.
10:  *
11:  * @author CakeSolutionGroup
12:  * @version 1.0 (1 May 2013)
13:  *
14:  */
15: public class View {
16:
17:     /**
18:      * The method'playerStatus' prints the current status of the player.
19:      *
20:      * @return null
21:      * @param sugarman
22:      */
23:     public void playerStatus(Sugarman sugarman) {
24:         System.out.println("This is your current sugar level: " + sugarman.getSugarlevel());
25:         System.out.println("This is your current location: " + sugarman.getLocation().getDescription());
26:     }
27:
28:     /**
29:      * The method 'printOutQuestion' prints the question and the adhering
30:      * choices.
31:      *
32:      * @return null
33:      * @param q
34:      */
35:     public void printOutQuestion(Question q) {
36:
37:
38:         System.out.println(q.question);
39:
40:         for (int i = 0; i < q.choices.size(); i++) {
41:
42:             Choice c = q.choices.get(i);
43:             System.out.println((i + 1) + ") " + c.choice);
44:
45:         }
46:     }
47:
```

```
48:     /**
49:      * the method prints out to ask if the user wants to stay in the location or
50:      * moving on to the next location.
51:      *
52:      * @param currentLocation
53:      * @param neighborLocation
54:      */
55:     public void printToMoveOrToStay(String currentLocation, String neighborLocation) {
56:         System.out.println("Do you wanna stay here in " + currentLocation
57:                 + " and answer questions or move on to " + neighborLocation
58:                 + "Type in 1 to stay or 2 to move on to next location or 3 to exit game");
59:
60:     }
61:
62:     /**
63:      * the method prints that there are no questions left in the location and
64:      * that the player should move to next location by pressing "2".
65:      */
66:     public void printNoQuestionsLeft() {
67:
68:         System.out.println("There are no questions left in this location! Move to the next location by choosing 2 :
 ");
69:     }
70:
71:     /**
72:      * prints to the user to enter answer
73:      */
74:     public void printEnterAnswer() {
75:
76:         System.out.println("Please enter the number of your answer: ");
77:
78:     }
79:
80:     /**
81:      * print out you got a cake, after the user's answer was correct.
82:      */
83:     public void printYouGotACake() {
84:
85:         System.out.println("Hurray you get a cake!");
86:
87:     }
88:
89:     /**
90:      * prints no cake for you, if the user's answer wasn't right.
91:      */
92:     public void printNoCake() {
93:
```

```
   94:         System.out.println("Sucker, you don't get a cake! HA!");
   95:     }
   96:
   97:     /**
   98:      * if the users input was anything else than between 1 to max, it prints out
   99:      * please enter an input from 1 to max.
  100:      *
  101:      * @param max
  102:      */
  103:     public void printRetypeInput(int max) {
  104:
  105:         System.out.println("Please, type an input from 1 to " + max);
  106:
  107:     }
  108:
  109:     /**
  110:      * when the user wins, print out this message.
  111:      */
  112:     public void printSugarmanWon() {
  113:
  114:         System.out.println("SPLASH! Hurray Sugarman won! He exploded and is now a happy part of all the cakes - JAY
!");
  115:     }
  116:
  117:     /**
  118:      * when the user loses, it prints out this message.
  119:      */
  120:     public void printSugarmanLost() {
  121:
  122:         System.out.println("Sorry Sugarman got too thin and got the SugarCold...and then he died...sorry.");
  123:     }
  124:
  125:     /**
  126:      * when there are not question left in all locations, it prints out this
  127:      * message.
  128:      */
  129:     public void printRanOutOfQuestions() {
  130:
  131:         System.out.println("Sorry you ran out of questions. Sugarman will run around crying until you Start over.")
;
  132:     }
  133:
  134:     /**
  135:      * when the user chose to exit the game, it prints out this
  136:      * message.
  137:      */
  138:
```

```
139:     public void printExitGame() {
140:
141:         System.out.println("You have exited the game. Bye bye!");
142:     }
143: }
```

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5: package cakesolutioncorrectversion;
 6:
 7: import java.util.List;
 8:
 9: /**
10:  * The interface 'World' creates a contract that the classes implementing 'World'
11:  * will contain a method 'getLocations'.
12:  * @author CakeSolutionGroup
13:  * @version 1.0 (1 May 2013)
14:  */
15: public interface World {
16:
17:     /** This method  will create a list of the Locations using the
18:      *  Class type 'Location'
19:      */
20:
21:     List<Location> getLocations();
22:
23:     public boolean isOutOfQuestions();
24:
25: }
26:
```