```
 1 package missioncontrol;
 2
 3 import java.util.Scanner;
 4
 5 /**
 6  * Class respresents a mission control station in a disaster site rescue
 7  * simulation. This class is responsible for creating the simulation objects
 8  * and facilitating the user's interaction with those objects.
 9  *
10  * @author Eric Darsow
11  */
12 public class MissionControl {
13
14     // total personnel available for deployment
15     final int totalStaff = 10;
16
17     // the DisasterSite object is stored in a member variable
18     // so all methods in MissionControl can access it
19     private static DisasterSite mainSite;
20
21     /**
22      * Program entry point--creates DiasterSite and
23      * Transfers control to the commander
24      * @param args no parameters needed
25      */
26     public static void main(String[] args) {
27         // create a DisasterSite object and store it in a class mem variable
28         mainSite = new DisasterSite();
29         // transfer execution control to this method
30         commanderControl();
31     }
32
33     /**
34      * Utility class for writing notices to the console.
35      * Uses a sleep method to simulate the communication process
36      * @param event the text about the event to be logged
37      */
38     public static void logEvent(String event){
39         // spacing only
40         System.out.println("");
41         // we need this try/catch thing for reasons we don't understand yet
42         try {
43             // wait 1 second
44             Thread.sleep(1000);
45             // print out whatever string was passed into the method
46             System.out.println("log entry: " + event);
47             Thread.sleep(1000);
48         } catch (InterruptedException ex) {
49             System.out.println(ex.toString());
50         }
51         System.out.println("");
52     } // close method
53
```

```
54      /**
55       * Coordinates all disaster recovery related events
56       */
57      public static void commanderControl(){
58          // stores user action for switch statement control
59          int action = 0;
60          // allow the commander to act until the exit signal is passed in
61          while(action != -1){
62              // transfer control to a method for gathering a single int
63              // from the Commander user and passing it back into this method
64              action = getCommanderAction();
65              // The commander's choice is processed by the switch statement
66              // if the user enters 1, then case 1 is executed.
67              switch (action) {
68                  // Write current vicitms to the log
69                  case 1: int result = mainSite.getCurrentVictimCount();
70                          logEvent("Current Victims: "+String.valueOf(result));
71                          break;
72                  // Assemble and dispatch crew
73                  case 2: dispatchCrew();
74                          break;
75              } // end switch
76          } // end while
77      } // close method
78
79
80      /**
81       * Utility method for gathering user input, i.e. the commander actions
82       * Prints out the options for an action and prepares it for return
83       * @return the user's selection
84       */
85      public static int getCommanderAction(){
86          System.out.println("*********************************");
87          System.out.println("MISSION CONTROL: COMMANDER ACTIONS:");
88          System.out.println("*********************************");
89
90          System.out.println("1. Write current victim count to the log");
91          System.out.println("2. Assemble and dispatch rescue Team");
92          System.out.println("3. Check mission status");
93          System.out.println("-1. Exit Program");
94          System.out.println("*********************************");
95          // prompt user
96          System.out.print("Enter desired action number: ");
97          Scanner scanner = new Scanner(System.in);
98
99          // get input from the user
100         int commanderChoice = scanner.nextInt();
101         // send the choice back tot he calling method
102         return commanderChoice;
103     } // close method
104
105     /**
106      * Simulates the formation and dispatching of the rescue crew
```

called from main()

call down to dispatchCrew

points to our ONE DisasterSite object

this method is only on DisasterSite objects

call each time through while loop

call from commanderControl()

call to nextInt() on Scanner object in Java Library

```
107      * to the DisasterSite. Handles crew creation.
108      */
109     public static void dispatchCrew() {
110         Scanner scanner = new Scanner(System.in);
111         // prompt user and get size
112         System.out.print("Enter a crew size to dispatch: ");
113         int crewSize = scanner.nextInt();
114
115         // build a new RescueTeam object to send to the DisasterSite
116         RescueTeam rescueTeam1 = new RescueTeam();
117         // set the team size on the RescueTeam object
118         rescueTeam1.setTeamSize(crewSize);
119
120         // Send the rescue team to make an attempt. The returned
121         // value from makeRescueAttempt() is the count of rescued victims
122         int numRescued = mainSite.makeRescueAttempt(rescueTeam1);
123         System.out.println("Rescue underway...");
124         // send the result to the log
125         logEvent("Number vicitims rescued: " + numRescued);
126
127     } // close method
128 } // close class
```