```
 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  * @author Eric Darsow
10  */
11 public class MissionControl {
12     // total personnel available for deployment
13     final int totalStaff = 10;
14     // the DisasterSite object is stored in a member variable
15     // so all methods in MissionControl can access it
16     private static DisasterSite mainSite;
17
18     /**
19      * Program entry point--creates DiasterSite and
20      * Transfers control to the commander
21      * @param args no parameters needed
22      */
23     public static void main(String[] args) {
24         // create a DisasterSite object and store it in a class member variable
25         mainSite = new DisasterSite();
26         // transfer execution control to this method
27         commanderControl();
28     } // close main
29     /**
30      * Utility class for writing notices to the console.
31      * Uses a sleep method to simulate the communication process
32      * @param event the text about the event to be logged
33      */
34     public static void logEvent(String event){
35         // spacing only
36         System.out.println("");
37         // we need this try/catch thing for reasons we don't understand yet
38         try {
39             // wait 1 second
40             Thread.sleep(1000);
41             // print out whatever string was passed into the method
42             System.out.println("log entry: " + event);
43             Thread.sleep(1000);
44         } catch (InterruptedException ex) {
45             System.out.println(ex.toString());
46         } // close try/catch
47         System.out.println("");
48     } // close method
49
50     /**
51      * Coordinates all disaster recovery related events
52      */
53     public static void commanderControl(){
54         // stores user action for switch statement control
55         int action = 0;
56         // allow the commander to act until the exit signal is passed in
57         while(action != -1){
58             // transfer control to a method for gathering a single int
59             // from the Commander user and passing it back into this method
```

```
60              action = getCommanderAction();
61              // The commander's choice is now processed by the switch statement
62              // if the user enters 1, then case 1 is executed.
63              switch (action) {
64                  // Write current vicitms to the log
65                  case 1: int result =
66                          mainSite.numTotalVictims - mainSite.numRescuedVictims;
67                          logEvent("Current Victims: " + String.valueOf(result));
68                          break;
69                  // Assemble and dispatch crew
70                  case 2: dispatchCrew();
71                          break;
72              } // end switch
73          } // end while
74      } // close method
75
76      /**
77       * Utility method for gathering user input, i.e. the commander actions
78       * Prints out the options for an action and prepares it for return
79       * @return the user's selection
80       */
81      public static int getCommanderAction(){
82          System.out.println("**********************************");
83          System.out.println("MISSION CONTROL: COMMANDER ACTIONS:");
84          System.out.println("**********************************");
85          System.out.println("1. Write current victim count to the log");
86          System.out.println("2. Assemble and dispatch rescue Team");
87          System.out.println("3. Check mission status");
88          System.out.println("-1. Exit Program");
89          System.out.println("**********************************");
90          System.out.print("Enter desired action number: ");
91          Scanner scanner = new Scanner(System.in);
92
93          // get input from the user
94          int commanderChoice = scanner.nextInt();
95          // send the choice back tot he calling method
96          return commanderChoice;
97      } // close method
98
99      /**
100      * Simulates the formation and dispatching of the rescue crew
101      * to the DisasterSite. Handles crew creation.
102      */
103     public static void dispatchCrew() {
104         Scanner scanner = new Scanner(System.in);
105         // prompt user and get size
106         System.out.print("Enter a crew size to dispatch: ");
107         int crewSize = scanner.nextInt();
108         // build a new RescueTeam object to send to the DisasterSite
109         RescueTeam rescueTeam1 = new RescueTeam();
110         // set the team size on the RescueTeam object
111         rescueTeam1.setTeamSize(crewSize);
112         // Send the rescue team to make an attempt. The returned
113         // value from makeRescueAttempt() is num of rescued victims
114         int numRescued = mainSite.makeRescueAttempt(rescueTeam1);
115         System.out.println("Rescue underway...");
116         logEvent("Number vicitims rescued: " + numRescued);
117     } // close method
118 } // close class
```