```java
import javax.swing.JOptionPane;
import javax.swing.JTextArea;

public class _9_4_FinalVersion {
	public static void main(String[] args) {
		/** HITS AND BULLSEYES                                        */
		/** COPYRIGHT 2011 DR BRUCE MARTIN RYAN.       */

		/** declarations */
		boolean debugMode;
		final int INTEGER_LENGTH = 3;
		final int NUMBER_OF_INTEGERS_FOR_2D_GAME = 3;
		String userName;

		/** declarations for menu */
		String fragment0 = ", please select one of the following options: \n\n";
		String fragment1 = "1 - View instructions\n";
		String fragment2 = "2 - Play 1-dimensional debug mode.\n";
		String fragment3 = "3 - Play 1-dimensional REAL mode\n";
		String fragment4 = "4 - Play 2-dimensional debug mode\n";
		String fragment5 = "5 - Play 2-dimensional REAL mode\n";
		String fragment6 = "6 - Exit";
		String menuText;
		String userChoiceString;
		int userChoice;

		/** code */

		/** Welcome the user, get its name. */
		userName = welcome(INTEGER_LENGTH);

		/** Set up text for menu. */
		menuText = userName + fragment0 + fragment1 + fragment2 + fragment3 + fragment4 + fragment5 + fragment6;

		/** Main menu is generated by a do/while loop. */
		do {
			/** Prompt user for choice. */
			userChoiceString = JOptionPane.showInputDialog(menuText);

			/** Convert user's choice into int. */
			userChoice = Integer.parseInt(userChoiceString);

			/** Use switch statement to call relevant method. */
			switch (userChoice) {
```

```
                        case  1:
                               displayInstructions(INTEGER_LENGTH, NUMBER_OF_INTEGERS_FOR_2D_GAME);
                               break;

                        case 2:
                               debugMode = true;
                               oneDGame(debugMode, INTEGER_LENGTH, userName);
                               break;

                        case 3:
                               debugMode = false;
                               oneDGame(debugMode, INTEGER_LENGTH, userName);
                               break;

                        case 4:
                               debugMode = true;
                        twoDGame(debugMode, INTEGER_LENGTH, userName, NUMBER_OF_INTEGERS_FOR_2D_GAME);
                               break;

                        case 5:
                               debugMode = false;
                        twoDGame(debugMode, INTEGER_LENGTH, userName, NUMBER_OF_INTEGERS_FOR_2D_GAME);
                               break;

                        case 6:
                               terminate();
                               break;

                        default:
                               JOptionPane.showMessageDialog(null, "You tried to make an illegal choice.", "INVALID INPUT",
JOptionPane.INFORMATION_MESSAGE);
                               break;
                    }

                } while (userChoice != 6);
                // end menu do loop

        } // end main

        /** User-defined methods. */

        /** This method welcomes the user and finds its name, which is returned as a String. */
        public static String welcome(int INTEGER_LENGTH) {
```

```
		/** declarations */
		String userName;
		String message = "Welcome to this wonderful game from RANDOM BOZO PRODUCTIONS.\n\nPlease enter your name.";

		/** code */
		/** Dialog to enter user's name - I WANT THIS TO HAVE A HEADLINE OTHER THAN 'input'!!! */
		userName = JOptionPane.showInputDialog(null, message);

		/** Return user's name, end method. */
		return userName;
	} // end welcome

	/** This method is simply the instructions to the user. */
	public static void displayInstructions(int INTEGER_LENGTH, int NUMBER_OF_INTEGERS_FOR_2D_GAME) {
		/** declarations */
		String instructionsText = "Welcome to another fine game from RANDOM BOZO UNLIMITED!\n\n" +
				"The program will generate a random " + INTEGER_LENGTH + "-digit number which does not contain any duplicate
digits.\n" +
				"Your misson, should you choose to accept it, is to work out the computer's number by entering a " + INTEGER_LENGTH +
"-digit number.\n" +
				"The program will check that your number has the right number of digits.\n" +
				"The program will also check that your number does not contain any duplicate digits.\n" +
				"(If your number fails either of these tests, this try will not count towards your total: we assume you have a dodgy
keyboard.)\n\n" +
				"After each round, the computer will tell you how many 'hits' and 'bullseyes' you have.\n" +
				"A hit is when one of your digits also occurs in the computer's number but in a different place.\n" +
				"For example, if the computer's number is 123 then the try 385 has one hit because the number 3 occurs but in a
different place.\n\n" +
				"A bullseye is when one of your digits occurs in the same place in the computer's number.\n" +
				"For example, if the computer's number is 123, the try 725 has one bullseye;\n" +
				"the number 2 is correct and occurs in the correct place.\n\n" +
				"Many tries will contain a mixture of hits and bullseyes.\n" +
				"For example, if the computer's number is 123 then the try 132 has one bullseye (for '1') and " +
				"two hits.\n('2' and '3' are correct digits but are not in the correct places.)\n\n" +
				"When you have deduced the computer's number, you will be told how many tries this took.\n" +
				"Don't be worried about ending up in Infinite Loop, Cupertino - you will have many chances to escape.\n\n\n" +
				"For REAL FUN, you can also play in 2 dimensions!\n" +
				"The program will generate " + NUMBER_OF_INTEGERS_FOR_2D_GAME + " random " + INTEGER_LENGTH + "-digit numbers, " +
				"each of which contains no duplicate digits.\n" +
				"Numbers may be duplicated on different rows, for example:\n" +
				" \t\t1  5  7\n\t\t8  2  5\n\t\t1  2  3\n\n" +
				"You then need to enter " + NUMBER_OF_INTEGERS_FOR_2D_GAME + " random " + INTEGER_LENGTH + "-digit numbers.\n" +
				"The program will check that each of your numbers has the correct number of unique digits.\n" +
				"(Again, we forgive dodgy keyboards.)\n\n" +
```

```
                        "After each round, the computer will tell you how many 'hits' and 'bullseyes' you have per row.\n" +
                        "For example, if the computer's numbers are as above and you try\n" +
                        "\t 1  8  9\n\t\t7  8  9\n\t\t1  3  6\nyou would score:\n" +
                        "\t    Row 1 - 0 hits, 1 bullseye\n" +
                        "\t    Row 2 - 1 hit, 0 bullseyes\n" +
                        "\t    Row 3 - 1 hit, 1 bulleye.\n\n\n" +
                        "DEBUG MODE\n" +
                        "For both 1-dimensional and 2-dimensional play, you can check that RANDOM BOZO UNLIMITED is not cheating you by
playing in debug mode.\n" +
                        "When this mode is selected, the computer's number(s) will be shown in the window where you enter your try.\n\n" ;

            /** code */
            /** Create a text area. */
            JTextArea textBox = new JTextArea(5, 6);

            /** Send text to output area. */
            textBox.setText(instructionsText);

            /** Display text. */
            JOptionPane.showMessageDialog(null, textBox, "INSTRUCTIONS", JOptionPane.INFORMATION_MESSAGE);
        } // end displayInstructions

        /** A method to run a complete 1-D game. Uses many of the following methods. */
        public static void oneDGame(boolean debugMode, int INTEGER_LENGTH, String userName) {

            /** declarations */
            int numberOfRounds;
            int [] random1DArray;
            boolean anotherRound;
            int user1DInputInteger;
            int [] user1DInputArray = new int [INTEGER_LENGTH];
            boolean correctNumberOfDigits;
            boolean uniqueDigits;
            boolean validTry;
            boolean wishToRetry;
            boolean needToRetry;
            int bullseyes = 0;
            int hits = 0;
            int maxBullseyes = INTEGER_LENGTH;

            /**code */
            /** Set number of rounds FOR CURRENT 1-D GAME to zero. */
            /** This could have been done in the declarations but seems neater here. */
            numberOfRounds = 0;
```

```
/** Get computer's random number as an array. */
random1DArray = getRandom1DArray(INTEGER_LENGTH);

/** Do/while loop to offer user unlimited rounds to find computer's number. */
do {
	/** Initialise for this round. */
	anotherRound = true;

	/** Inner do/while loop to offer user unlimited tries to enter valid input. */
	/** Invalid tries do not count towards total number of attempts. */
	do {
		/** Initialise variables to do with exiting this loop. */
		wishToRetry = true;
		needToRetry = true;

		/** Get user try as an integer. */
		user1DInputInteger = user1DInputInteger(INTEGER_LENGTH, debugMode, random1DArray);

		/** Get user try as a 1-D array. */
		user1DInputArray = user1DInputArray(INTEGER_LENGTH, user1DInputInteger);

		/** Validate user input. */
		/** The two criteria are tested together so that the user input has to path both tests simultaneously to be valid. */
		/** Otherwise it might be possible for a number that has the right number of non-unique digits or */
		/** the wrong number of unique digits to slip through. */

			/** Validate correct number of digits. */
			correctNumberOfDigits = correctNumberOfDigits(INTEGER_LENGTH, user1DInputInteger);

			/** Validate uniqueness of digits. */
			uniqueDigits = uniqueDigits(INTEGER_LENGTH, user1DInputArray);

			/** Put validation tests together into one boolean variable. */
			if (correctNumberOfDigits == true && uniqueDigits == true) {
				validTry = true;
			}
			else validTry = false;
			//end if

			/** If the try was invalid, report why and offer user another try to enter valid input, if necessary. */
			/** If user doesn't want another try, we escape this inner do/while loop. */
			/** Of course, we also escape this do/while loop if the user's try is valid! */
			if (validTry == false) {
```

```
                        wishToRetry = wishToRetry(correctNumberOfDigits, uniqueDigits, INTEGER_LENGTH);
                  } // end if

            /** Put potential reasons for retrying into one boolean variable to decide whether to stay in or finish inner do/while
loop. */
            if (validTry == true) {
                  needToRetry = false;
            } // end if

            if (validTry == false && wishToRetry == true) {
                  needToRetry = true;
            } // end if

            if (validTry == false && wishToRetry == false) {
                  needToRetry = false;
            } // end if

      } while (needToRetry == true);

      /** Break back to main menu if user doesn't want another try. */
      if (wishToRetry == false) {
            break;
      }

      /** Increment number of rounds. */
      numberOfRounds = numberOfRounds + 1;

      /** Find number of bullseyes. */
      bullseyes = numberOfBullseyes(INTEGER_LENGTH, random1DArray, user1DInputArray);

      /** Find number of hits. */
      hits = numberOfHits(INTEGER_LENGTH, random1DArray, user1DInputArray);

      /** Display hits and bullseyes for this round, find out if user wants another round.                */
      /**    If number of bullseyes = max possible, report = 'congratulations! You took x rounds.   */
      /**          Otherwise just report hits and bullseyes, asking user if it wants another round       */

      /** BAH! This was a PITA - just calling the method doesn't work. Why did it take me a day to work out I needed 'anotherTry' =
<method>? */
      anotherRound = display1DScore(bullseyes, hits, INTEGER_LENGTH, numberOfRounds, userName, maxBullseyes);

      /** Finally, if user has got max number of bulleyes, we can break back to main menu. */
      if (bullseyes == maxBullseyes) {
            break;
```

```
			} // end if

		} while (anotherRound == true);
		// end do/while loop

} // end 1DGame

/** This method calls the 'ranDigit()' method sufficient times to populate a 1-D array (of length INTEGER_LENGTH) with random digits. */
public static int[] getRandom1DArray(int INTEGER_LENGTH) {

		/** declarations */
		int [] random1DArray = new int [INTEGER_LENGTH];
		boolean uniqueRandomDigits;

		/** code */
		/** Populate array with unique random digits. The first digit cannot be zero: this is enforced by the while condition. */
		do {
			for (int xWidthCoord = 0; xWidthCoord < INTEGER_LENGTH; xWidthCoord++) {
				random1DArray[xWidthCoord] = ranDigit();
			} // end for

			/** Ensure that digits are unique. */
			uniqueRandomDigits = uniqueDigits(INTEGER_LENGTH, random1DArray);
		} while (uniqueRandomDigits == false || random1DArray[0] == 0);
		// end do

		/** Return result, end method. */
		return random1DArray;
} // end getRandom1DArray

/** This method uses the standard Math class to generate a random number, which is then rounded up to an integer. */
public static int ranDigit() {
		/** declarations */
		double randomNumber;
		int randomDigit;

		/** code */
		/** Generate random number. */
		randomNumber = Math.random();

		/** Make this a digit between 0 and 9. */
		randomDigit = (int) (randomNumber * 10);

		/** Note: had I needed to exclude zero from all digits (I only need to exclude it from the first digit and that's taken care of in
```

```
'getRandomArray()' */
            /** [which might be inefficient but it seems silly to write two random digit generators]), this would have been: */
            /** do {                                                                                  */
            /**    randomNumber = Math.random();                          */
            /**    randomDigit = (int) (randomNumber * 10);       */
            /** } while (randomDigit == 0);                               */
            /** // end do                                                                  */

            /** Return result, end method. */
            return randomDigit;
      } // end ranDigit

      /** This method tests all elements of a 1-D integer array for uniqueness and returns a boolean result. */
      /** Obviously, the array width has to be 9 or less because we're using denary numbers. */
      public static boolean uniqueDigits(int INTEGER_LENGTH, int [] oneDArray) {
            /** declarations */
            boolean valid = true;

            /** code */
            for (int toBeTestedCounter = 1; toBeTestedCounter < INTEGER_LENGTH; toBeTestedCounter++) {

                  /** Test for uniqueness: test 2nd digit against 1st, 3rd digit against 2nd and 1st, and so on. */
                  for (int testCounter = 0; testCounter < toBeTestedCounter; testCounter++) {
                        if (oneDArray[toBeTestedCounter] == oneDArray[testCounter]) {
                        valid = false;
                        } // end if
                  } // end for
            } // end for

            /** Return result, end method. */
            return valid;
      } // end uniqueDigits

      /** This method asks for a String, then converts it to an integer. */
      /** There is no error-trapping for non-numeric input! */
      public static int user1DInputInteger(int INTEGER_LENGTH, boolean debugMode, int [] random1DArray) {

            /** declarations */
            String inputMessage;
            String computersNumberString;
            String debugMessage = "The random number you're trying to match is ";
            String user1DInputString;
            int user1DInputInteger;
```

```
        /** code */
        /** Ask user for input. */
        inputMessage = "Please enter a " + INTEGER_LENGTH + "-digit number which has unique digits.\n" +
                       "By the way, an initial zero is invalid. (For example, 045 is really just 45.)";

        /** If user has chosen debug mode, we need to add computer's number to the input dialog. */
        if (debugMode == true) {

               /** Construct String for computer's number. */
               /** Initialise this string here in case we go around do loop several times. */
               computersNumberString = "";

               for (int debugCounter = 0; debugCounter < INTEGER_LENGTH; debugCounter++) {
                      computersNumberString = computersNumberString + random1DArray[debugCounter] + " ";
               } // end for

               inputMessage = inputMessage + "\n\n" + debugMessage + computersNumberString;
        } //end if

        /** If user has not chosen debug mode, just ask for input. */
        user1DInputString = JOptionPane.showInputDialog(inputMessage);

        /** Convert user input into integer. */
        user1DInputInteger = Integer.parseInt(user1DInputString);

        /** Return result, end method. */
        return user1DInputInteger;
} // end user1DInputInteger

/** This method puts the individual digits of an integer into an away of length INTEGER_LENGTH, then returns this array.      */
/** Because the value of 'a' in the integer 'abcde' is 'a' * 10^^4, we can divide 'abcde' by 10^^4 to get 'a':                 */
/** the result will be rounded down, because we're dealing with Java integers, to 'a'.
        */
/** We know that the power of 10 to divide by will be one less than INTEGER_LENGTH.
        */
/** This method uses the 'Math.pow' function to find the actual value of this factor.
        */
/** We can then subtract 'a' * 10^^4 from abcde to begin the process of finding 'b'...
        */
public static int[] user1DInputArray(int INTEGER_LENGTH, int user1DInputInteger) {

        /** declarations */
        int factor;
        int [] user1DArray = new int [INTEGER_LENGTH];
```

```
            int [] Remainder1DArray = new int [INTEGER_LENGTH];

            /** code */
                  /** Find individual digits of user input. */

                        /** The first digit is simply the complete user input divided by 10 ^^ (array-width - 1), because ints are always
rounded down. */
                        factor = (int) Math.pow(10,(INTEGER_LENGTH - 1));
                        user1DArray[0] =  user1DInputInteger / factor;

                        /** Set up remainder of user input for loop for subsequent digits. */
                        Remainder1DArray[0] = user1DInputInteger - (user1DArray[0] * factor);

                        /** Subsequent digits are remainder divided by decreasing powers of 10. */
                        for (int userArrayCounter = 1; userArrayCounter < INTEGER_LENGTH; userArrayCounter++) {
                              factor = (int) Math.pow(10,(INTEGER_LENGTH - userArrayCounter - 1));
                              user1DArray[userArrayCounter] =  Remainder1DArray[(userArrayCounter - 1)] / factor;
                              Remainder1DArray[userArrayCounter] = Remainder1DArray[(userArrayCounter - 1)] - (user1DArray[userArrayCounter] *
factor);
                        } // end for

                        /** If I were only considering 3-digit numbers, the following would have worked       */
                        /** userArray[0] =  userInput / 100;
      */
                        /** userArray[1] =  (userInput - (userArray[0] * 100)) / 10;                                  */
                        /** userArray[2] =  (userInput - (userArray[0] * 100) - (userArray[1] * 10)) / 1;     */

            /** Return result, end method. */
            return user1DArray;
      } // end user1DInputArray

      /**This method tests whether an integer is inclusively between two bounds calculated from the required array width and returns a boolean
result. */
      public static boolean correctNumberOfDigits(int INTEGER_LENGTH, int user1DInputInteger) {

            /** declarations */
            int userInputMin;
            int userInputMax;
            boolean correctNumberOfDigits;

            /** code */
            /** Calculate bounds. */
            userInputMin = (int) (Math.pow(10,(INTEGER_LENGTH - 1))-1);
            userInputMax = (int) (Math.pow(10,INTEGER_LENGTH)-1);
```

```
		if (user1DInputInteger >= userInputMin && user1DInputInteger <= userInputMax) {
			correctNumberOfDigits = true;
		}
		else correctNumberOfDigits = false;
		// end if

		/** Return result, end method. */
		return correctNumberOfDigits;
	} // end correctNumberOfDigits

	/** This method uses the value of two booleans to choose the appropriate feedback. */
	/** The user is asked to make a YES/NO choice: if Yes (= 0) is chosen, true is returned. */
	public static boolean wishToRetry(boolean correctNumberOfDigits, boolean uniqueDigits, int INTEGER_LENGTH) {

		/** declarations */
		String message;
		String escapeFragment = "\n\n(Choosing 'no' will take you back to the main menu.)";
		int wishToRetryInt = 0;
		String invalidInputHeadline = "INVALID INPUT";
		boolean wishToRetryBoolean;

		/** code */
		if (correctNumberOfDigits == false && uniqueDigits == true) {
			message = "Your number didn't have " + INTEGER_LENGTH + " digits.\nWould you like another try?" + escapeFragment;
			wishToRetryInt = JOptionPane.showConfirmDialog(null, message, invalidInputHeadline, JOptionPane.YES_NO_OPTION);
		} // end if

		if (correctNumberOfDigits == true && uniqueDigits == false) {
			message = "For a thing to be unique, it must be different to all other relevant things.\n" +
				"Now you know this, would you like another try?" + escapeFragment;
			wishToRetryInt = JOptionPane.showConfirmDialog(null, message, invalidInputHeadline, JOptionPane.YES_NO_OPTION);
		} // end if

		if (correctNumberOfDigits == false && uniqueDigits == false) {
			message = "Your number didn't have " + INTEGER_LENGTH + " digits.\n\n" +
				"Further, for a thing to be unique, it must be different to all other relevant things.\n" +
				"Now you know this, would you like another try?" + escapeFragment;
			wishToRetryInt = JOptionPane.showConfirmDialog(null, message, invalidInputHeadline, JOptionPane.YES_NO_OPTION);
		} // end if

		/** Convert integer response to boolean because it's easier to work with 'true' and 'false' than to remember whether '1' means 'yes'
or 'no'! */
		/** YES = 0, NO = 1. */
```

```
	/** Also, I want to work with booleans in the method that calls this one. */
	if (wishToRetryInt == 0) {
		wishToRetryBoolean = true;
	}
	else wishToRetryBoolean = false;
	// end if

	/** Return result, end method. */
	return wishToRetryBoolean;
} // end wishToRetry

/** This method returns the number of bullseyes (user has entered a correct digit in the correct place) as an integer. */
public static int numberOfBullseyes(int INTEGER_LENGTH, int []user1DArray, int [] random1DArray) {

	/** declarations */
	int bullseyes = 0;

	/** code */
	for (int bCounter = 0; bCounter < INTEGER_LENGTH; bCounter++) {
		if(user1DArray[bCounter]== random1DArray[bCounter]) {
			bullseyes = bullseyes + 1;
		} // end if
	} // end for

	/** Return result, end method. */
	return bullseyes;
} // end numberOfBullseyes

/** This method returns the number of hits (user has entered a correct digit but in the wrong place) as an integer. */
public static int numberOfHits(int INTEGER_LENGTH, int [] random1DArray, int [] user1DArray) {
	/** Why do I make life difficult for myself by going for the general case?                                    */
	/** We need to NOT test computerArray[x] = userArray[x] because that's testing for bullseyes.*/
	/** But we do need to test all possible computerArray[x] = userArray[y].                                 */
	/** Therefore two halves of test loop: above and below x.                                                                */

	/** declarations */
	int testee;
	int omit;
	int hits = 0;

	/** code */
	for (int hitCounter1 = 0; hitCounter1 < INTEGER_LENGTH; hitCounter1++) {
		testee = random1DArray[hitCounter1];
		omit = hitCounter1;
```

```
                        /** Test 'below' current potential bullseye position. */
                        for (int hitCounter2 = 0; hitCounter2< omit; hitCounter2++) {
                              if (user1DArray[hitCounter2] == testee) {
                                    hits = hits + 1;
                              } // end if
                        } // end for

                        /** Test 'above' current potential bullseye position. */
                        for (int hitCounter3 = (omit + 1); hitCounter3 < INTEGER_LENGTH; hitCounter3++) {
                              if (user1DArray[hitCounter3] == testee) {
                                    hits = hits + 1;
                              } // end if
                        } // end for
                  } // end for

                  /** Return result, end method. */
                  return hits;
            } // end numberOfHits

            /** This method displays the users's score for the most recent 1-D round, calling the 'congratulate user()' method if the user got the
maximum number of bullseyes. */
            /** It uses a YES/NO dialog box to ask if the user would like another round. */
            /** The result of the YES/NO (YES = 0, NO = 1) is converted to a boolean - the boolean is returned. */
            public static boolean display1DScore(int bullseyes, int hits, int INTEGER_LENGTH, int numberOfRounds, String userName, int
maxBullseyes) {

                  /** declarations */
                  String fragment1 = "You scored ";
                  String fragment2;
                  String fragment3;
                  String fragment4 = "Would you like another round?";
                  String normalHeadline = "Your score";
                  String headline = "";
                  String output = "";
                  int anotherRoundInt = 0;
                  boolean anotherRoundBoolean;

                  /** code */
                  /** Sort singular/plural issues in report. */
                  if (bullseyes == 1) {
                        fragment2 = " bullseye and ";
                  }
                  else fragment2 = " bullseyes and ";
```

```
        // end if
        if (hits == 1) {
                fragment3 = " hit.";
        }
        else fragment3 = " hits.";
        // end if

        /** If user got maximum number of bullseyes, congratulate user. */
        if (bullseyes == INTEGER_LENGTH) {
                congratulateUser(INTEGER_LENGTH, numberOfRounds, userName, maxBullseyes);
        }
        /** If user didn't get maximum number of bullseyes, report score and offer another round. */
        else {
                headline = normalHeadline;
                output = fragment1 + bullseyes + fragment2 + hits + fragment3 + "\n\n" + fragment4;
                anotherRoundInt = JOptionPane.showConfirmDialog(null, output, headline, JOptionPane.YES_NO_OPTION);
        } // end if

        /** Convert user's response to boolean. */
        if (anotherRoundInt == 0) {
                anotherRoundBoolean = true;
        }
        else anotherRoundBoolean = false;

        /** Return whether the user wants another round, end method. */
        return anotherRoundBoolean;
} // end display1DScore

/** This method displays a dialog showing how many valid rounds the user took to get the maximum number of bullseyes. */
public static void congratulateUser(int INTEGER_LENGTH, int numberOfRounds, String userName, int maxBullseyes) {

        /** declarations */
        String fragment1 = ", you scored the maximum of ";
        String fragment2 = " bullseyes!";
        String fragment3 = "\n\nThis took only ";
        String fragment4;
        String fragment5 = "\n(Tries with non-unique digits and incorrect numbers of digits\ndid not count towards your score.)";
        String finishHeadline = "CONGRATULATIONS";
        String output;

        /** code */
        /** Sort grammar. */
        if (numberOfRounds == 1) {
                fragment4 = " round.";
```

```
        }
        else fragment4 = " rounds.";
        // end if

        /** Generate output. */
        output = userName + fragment1 + maxBullseyes + fragment2 + fragment3 + numberOfRounds + fragment4 + "\n" + fragment5;
        JOptionPane.showMessageDialog(null, output, finishHeadline, JOptionPane.INFORMATION_MESSAGE);

}// end congratulateUser

/** A method to run a complete 2-D game. Uses many of the following methods. */
/** The major difference here is that we are now dealing with 2-D arrays: each row represents one integer. */
public static void twoDGame(boolean debugMode, int INTEGER_LENGTH, String userName, int NUMBER_OF_INTEGERS_FOR_2D_GAME) {
        /** declarations */
        int numberOfRounds;
        int [] [] random2DArray;
        boolean anotherRound;
        boolean wishToRetry2D;
        boolean needToRetry2D;
        int [] [] user2DInputArray = new int [INTEGER_LENGTH] [NUMBER_OF_INTEGERS_FOR_2D_GAME];
        boolean [] correctNumberOf2DdigitsArray = new boolean [NUMBER_OF_INTEGERS_FOR_2D_GAME];
        boolean [] uniqueDigits2DArray = new boolean [NUMBER_OF_INTEGERS_FOR_2D_GAME];
        String inputMessageErrorString;
        boolean correctNumberOfDigits;
        boolean uniqueDigits;
        boolean validTry;
        int [] bullseyesArray;
        int bullseyes = 0;
        int [] hitsArray;
        int hits = 0;

        int maxBullseyes = INTEGER_LENGTH * NUMBER_OF_INTEGERS_FOR_2D_GAME;

        /**code */
        /** Set number of tries FOR CURRENT 2-D GAME to zero. */
        /** This could have been done in the declarations but seems neater here. */
        numberOfRounds = 0;

        /** Get computer's random numbers as a 2-D array. */
        random2DArray = getRandom2DArray(INTEGER_LENGTH, NUMBER_OF_INTEGERS_FOR_2D_GAME);

        /** Do/while loop to offer user unlimited rounds to find computer's number. */
        do {
                /** Initialise for this round. */
```

```
                anotherRound = true;

                /** Inner do/while loop to offer user unlimited tries to enter valid input. */
                /** Invalid tries do not count towards total number of attempts. */
                do {
                        /** Initialise variables to do with exiting this loop. */
                        wishToRetry2D = true;
                        needToRetry2D = true;

                        /** Get user tries as a 2-D array.
                                                */
                        /** Rather than interrupt the user's concentration with multiple interruptions due to invalid data,
        */
                        /** we save up all the validation until we have all the raw data.
                           */
                        /** Also, because we are programming for the general case (i.e. NUMBER_OF_INTEGERS_FOR_2D_GAME might be changed),
                        */
                        /** we cannot store integer 1, 2, 3... (or even a set of 1-D arrays) because we don't know how many we will need. */
                        /** All we know is that we need NUMBER_OF_INTEGERS_FOR_2D_GAME rows to store the user input.
                                */
                        /** Hence 2-D input must go straight into a 2-D array.
                                        */
                        user2DInputArray = user2DInputArray(INTEGER_LENGTH, NUMBER_OF_INTEGERS_FOR_2D_GAME, debugMode, random2DArray);

                        /** Validate user input. */
                        /** The two criteria are tested together so that the user input has to path both tests simultaneously to be valid. */
                        /** Otherwise it might be possible for a number that has the right number of non-unique digits or
             */
                        /** the wrong number of unique digits to slip through.
                                        */

                              /** Validate correct number of digits in each row.
                                                      */
                              /** Because we have to consider several numbers, we store info on this facet of validity of each number as an
element       */
                              /** in a 1-D array of booleans.
                                                          */
                              correctNumberOf2DdigitsArray = correctNumberOf2DdigitsArray(NUMBER_OF_INTEGERS_FOR_2D_GAME, INTEGER_LENGTH,
user2DInputArray);

                              /** Validate uniqueness of digits in each row. */
                              /** As with the number-of-digits validation, we store the validity of each bit of user input in a 1-D array of
booleans. */
                              uniqueDigits2DArray = uniqueDigits2DArray(NUMBER_OF_INTEGERS_FOR_2D_GAME, INTEGER_LENGTH, user2DInputArray);
```

```
                        /** Put validation tests together into one boolean variable.                                    */
                        /** Hooray - I can use one method to achieve two things!                                        */
                              /** First, if any row had an incorrect number of digits, the input is invalid.  */
                              correctNumberOfDigits = ifAnyElementIsFalseWholeArrayIsFalse(correctNumberOf2DdigitsArray,
NUMBER_OF_INTEGERS_FOR_2D_GAME);

                              /** Then, if any row had non-unique digits, the input is invalid. */
                              uniqueDigits = ifAnyElementIsFalseWholeArrayIsFalse(correctNumberOf2DdigitsArray,
NUMBER_OF_INTEGERS_FOR_2D_GAME);

                              /** Finally, the input must have passed both tests to be valid. */
                              if (correctNumberOfDigits == true && uniqueDigits == true) {
                                    validTry = true;
                              }
                              else validTry = false;
                              //end if

                        /** If the try was invalid, report why and offer user another try to enter valid input, if necessary.     */
                        /** If user doesn't want another try, we escape this inner do/while loop.
            */
                        /** Of course, we also escape this do/while loop if the user's try is valid!
            */
                        if (validTry == false) {

                              /** First, create input-error message String. */
                              inputMessageErrorString = inputMessageErrorString(correctNumberOf2DdigitsArray, uniqueDigits2DArray,
NUMBER_OF_INTEGERS_FOR_2D_GAME);

                              /** Then ascertain user's response. */
                              wishToRetry2D = wishToRetry2D(inputMessageErrorString, userName);

                        } // end if

                  /** Put potential reasons for retrying into one boolean variable to decide whether to stay in or finish inner do/while
loop. */
                  if (validTry == true) {
                        needToRetry2D = false;
                  } // end if

                  if (validTry == false && wishToRetry2D == true) {
                        needToRetry2D = true;
                  } // end if
```

```
		if (validTry == false && wishToRetry2D == false) {
			needToRetry2D = false;
		} // end if

	} while (needToRetry2D == true);

	/** Break back to main menu if user doesn't want another try. */
	if (wishToRetry2D == false) {
		break;
	}

	/** Increment number of rounds. */
	numberOfRounds = numberOfRounds + 1;

	/** Find number of bullseyes. If it's the maximum number we can move straight to the end of the game! */
		/** Firstly, get 1-D array containing number of bullseyes in each row of user's input. */
		bullseyesArray = getBullseyesArray (random2DArray, user2DInputArray, NUMBER_OF_INTEGERS_FOR_2D_GAME, INTEGER_LENGTH);

		/** Next, find total number of bulleyes. */
		bullseyes = 0;
		for (int bullseyesCounter = 0; bullseyesCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; bullseyesCounter++) {
			bullseyes = bullseyes + bullseyesArray[bullseyesCounter];
		} // end for

		/** If user got maximum number of bullseyes, congratulate it and return to main menu. */
		if (bullseyes == maxBullseyes) {
			congratulateUser(INTEGER_LENGTH, numberOfRounds, userName, maxBullseyes);
			break;
		} // end if

	/** If user didn't get maximum number of bullseyes, move on to finding number of hits. */
	/** Then report its score for this round and offer it another round. */

		/** Firstly get 1-D array containing number of hits in each row of user's input. */
		hitsArray = getHitsArray (random2DArray, user2DInputArray, NUMBER_OF_INTEGERS_FOR_2D_GAME, INTEGER_LENGTH);

		/** Next, find total number of hits. */
		hits = 0;
		for (int hitsCounter = 0; hitsCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; hitsCounter++) {
			hits = hits + hitsArray[hitsCounter];
		} // end for

	/** Display hits and bullseyes for this round, find out if user wants another round. */
		anotherRound = display2DScore(NUMBER_OF_INTEGERS_FOR_2D_GAME, bullseyesArray, hitsArray, bullseyes, hits,
```

```
INTEGER_LENGTH, numberOfRounds, userName);

                    /** Finally, if user has got max number of bulleyes, we can break back to main menu. */
                    // if (bullseyes == maxBullseyes) {
                    //     break;
                    // } // end if

             } while (anotherRound == true);
             // end do/while loop

      } // end twoDGame

      /** This method calls the 'getRandom1DArray()' method sufficient times to populate a 2-D array with validated random numbers. */
      /** This method is reused to ensure that all first digits are zero. */
      public static int [] [] getRandom2DArray(int INTEGER_LENGTH, int NUMBER_OF_INTEGERS_FOR_2D_GAME) {

             /** declarations */
             int [] [] random2DArray = new int [INTEGER_LENGTH] [NUMBER_OF_INTEGERS_FOR_2D_GAME];
             int [] Random1DArray;

             /** code */
             for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
                    Random1DArray = getRandom1DArray(INTEGER_LENGTH);
                    for (int widthCounter = 0; widthCounter < INTEGER_LENGTH; widthCounter++) {
                           random2DArray[widthCounter][heightCounter] = Random1DArray[widthCounter];
                    } // end for
             } //end for

             /** Return result, end method. */
             return random2DArray;
      } // end getRandom2DArray

      /** This method tells the user to input the required amount of integers, and returns the digits of these integers as elements in a 2-D
array. */
      public static int [] [] user2DInputArray(int INTEGER_LENGTH, int NUMBER_OF_INTEGERS_FOR_2D_GAME, boolean debugMode, int [] []
random2DArray) {
             /** declarations */

                    /** declarations for getting user input as integers */
                    int [] [] user2DInputArray = new int [INTEGER_LENGTH][NUMBER_OF_INTEGERS_FOR_2D_GAME];
                    int rowCounter;
                    String inputMessage;
                    String inputMessageFragment1 = "Your try, row ";
                    String inputMessageFragment2 = "Please enter a " + INTEGER_LENGTH + "-digit number which has unique digits.\n" +
```

```
      "By the way, an initial zero is invalid. (For example, 045 is really just 45.)";
String computersNumberString;
String debugMessage = "The random numbers you're trying to match are:\n";
String userInputString;
int userInputInteger;

/** declarations for converting user input integers into rows of a 2D-array */
int factor;
int [] [] Remainder2DArray = new int [INTEGER_LENGTH] [NUMBER_OF_INTEGERS_FOR_2D_GAME];

/** code */
/** Get user input as rows of a 2-D array. */
for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {

      /** We need to translate between human (first item = item 1) and Java (first item = item 0). */
      rowCounter = heightCounter + 1;
      inputMessage = inputMessageFragment1 + rowCounter + "\n" + inputMessageFragment2;

      /** If user has chosen debug mode, we need to add this to the input dialog. */
      if (debugMode == true) {

            /** A for loop to construct construct the String to display the computer's numbers. */
            /** We (re)initialise this string here in case we go around the loop several times. */
            computersNumberString = "";

            for (int yCoord = 0; yCoord < NUMBER_OF_INTEGERS_FOR_2D_GAME; yCoord++) {
                  for (int xCoord = 0; xCoord < INTEGER_LENGTH; xCoord++) {
                        computersNumberString = computersNumberString + random2DArray [xCoord] [yCoord] + " ";
                  } // end for
                  computersNumberString = computersNumberString + "\n";
            } // end for

            inputMessage = inputMessage + "\n\n" + debugMessage + computersNumberString;
      } //end if

      /** Ask user for input. */
      userInputString = JOptionPane.showInputDialog(inputMessage);

      /** Parse user input into an integer. */
      userInputInteger = Integer.parseInt(userInputString);

      /** Put the digits of that integer into the appropriate row of a 2-D array. */
      /** This is based on method for putting the digits of an integer into a 1-D array, */
      /** but has the added complication that we need to ensure the digits go into the correct row of the 2-D array. */
```

```
                        /** The first digit of a row is simply the complete user input divided by 10 ^^ (array-width - 1), because ints are
always rounded down. */
                        factor = (int) Math.pow(10,(INTEGER_LENGTH - 1));
                        user2DInputArray[0][heightCounter] =  userInputInteger / factor;

                        /** Set up remainder of user input for loop for subsequent digits. */
                        Remainder2DArray[0][heightCounter] = userInputInteger - (user2DInputArray[0][heightCounter] * factor);

                        /** Subsequent digits are remainder divided by decreasing powers of 10. */
                        for (int widthCounter = 1; widthCounter < INTEGER_LENGTH; widthCounter++) {
                              factor = (int) Math.pow(10,(INTEGER_LENGTH - widthCounter - 1));
                              user2DInputArray[widthCounter][heightCounter] = Remainder2DArray[(widthCounter - 1)][heightCounter] / factor;
                              Remainder2DArray[widthCounter][heightCounter] = Remainder2DArray[(widthCounter - 1)][heightCounter] -
(user2DInputArray[widthCounter][heightCounter] * factor);
                        } // end for

                  } // end for loop that gets individual rows of user input

                  /** Return result, end method. */
                  return user2DInputArray;
            } // end user2DInputArray

      /** Given the number of rows, this method adds the values of in each row of a 2-D array of integers, and stores these values in a 1-D
array which has */
      /** 1 element per row of the original 2-D array, thus recreating the user's original integers. */
      /** It then uses an existing method 'correctNumberOfDigits()' to find whether each element has the correct number of elements. */
      /** The result of these tests is returned as a 1-D array of booleans. */
      public static boolean [] correctNumberOf2DdigitsArray(int NUMBER_OF_INTEGERS_FOR_2D_GAME, int INTEGER_LENGTH, int [] []
user2DInputArray) {

            /** declarations */
            int rowSum;
            int [] recreateuserIntegerArray = new int [NUMBER_OF_INTEGERS_FOR_2D_GAME];
            boolean [] correctNumberof2DdigitsArray = new boolean [NUMBER_OF_INTEGERS_FOR_2D_GAME];
            double factor;

            /** code */
            /** Recreate user's original integers. */
            /** BAH - this seems inefficient! */
            for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {

                  /** Initialise the variable used to add up the digits. */
                  rowSum = 0;
```

```
                for (int widthCounter = 0; widthCounter < INTEGER_LENGTH; widthCounter++) {
                        factor = (int) Math.pow(10,(INTEGER_LENGTH - widthCounter - 1));
                        rowSum = rowSum + (int) (user2DInputArray[widthCounter][heightCounter] * factor);
                } // end for
                recreateuserIntegerArray[heightCounter] = rowSum;

        } // end for

        /** Send each element of resultant 1-D array for testing. */
        for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
                correctNumberof2DdigitsArray[heightCounter] = correctNumberOfDigits(INTEGER_LENGTH, recreateuserIntegerArray[heightCounter]);
        } // end for

        /** Return result, end method. */
        return correctNumberof2DdigitsArray;
        } // end correctNumberOf2DdigitsArray

    /** Given a 2-D array of integers, this method uses an existing method to test whether all the digits in each row are unique. */
    /** The result of these tests is returned as a 1-D array of booleans. */
    /** This method calls the 'uniqueDigits()' method sufficient times to test the uniqueness of the digits on each row of a 2-D integer
array. */
    public static boolean [] uniqueDigits2DArray(int NUMBER_OF_INTEGERS_FOR_2D_GAME, int INTEGER_LENGTH, int [] [] user2DInputArray) {

        /** declarations */
        int [] IndividualRowArray = new int [NUMBER_OF_INTEGERS_FOR_2D_GAME];
        boolean [] uniqueDigits2DArray = new boolean [NUMBER_OF_INTEGERS_FOR_2D_GAME];

        /** code */
        /** In turn, test whether the digits in each row are unique. */
        for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
                for (int widthCounter = 0; widthCounter < INTEGER_LENGTH; widthCounter++) {
                        IndividualRowArray[widthCounter] = user2DInputArray[widthCounter][heightCounter];
                } // end width for loop
                uniqueDigits2DArray[heightCounter] = uniqueDigits(INTEGER_LENGTH, IndividualRowArray);
        } // end height for loop

        /** Return result, end method. */
        return uniqueDigits2DArray;
        } // end uniqueDigits2DArray

    /** This method analyses the rows of a boolean 2-D array (width 2) to generate an error message String. */
    public static String inputMessageErrorString(boolean [] correctNumberOf2DdigitsArray, boolean [] uniqueDigits2DArray, int
NUMBER_OF_INTEGERS_FOR_2D_GAME) {
```

```
	/** declarations */
	int rowCounter;
	String digitsFragment;
	String uniquenessFragment;
	String userWishFragment = "Would you like to try again?\nChoosing 'no' will take you back to the main menu.";
	String inputMessageErrorString = "";

	/** code */
	for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
		/**Initialise variables: assume the entry was OK. */
		digitsFragment = "";
		uniquenessFragment = "";
		rowCounter = heightCounter + 1;

		/** Then report if the entry was NOT OK. */
		if (correctNumberOf2DdigitsArray[heightCounter] == false) {
			digitsFragment = "Your entry for row " + rowCounter + " had too many or too few digits.\n";
		} // end if

		/** If both errors are present, the statements should be grammatically linked. */
		if (uniqueDigits2DArray[heightCounter] == false && correctNumberOf2DdigitsArray[heightCounter] == false) {
			uniquenessFragment = "Your entry for row " + rowCounter + " also had non-unique digits.\n";
		} // end if

		if (uniqueDigits2DArray[heightCounter] == false && correctNumberOf2DdigitsArray[heightCounter] == true) {
			uniquenessFragment = "Your entry for row " + rowCounter + " had non-unique digits.\n";
		} // end if

		/** Put invalidity fragments together. */
		inputMessageErrorString = inputMessageErrorString + digitsFragment + uniquenessFragment + "\n";
	} // end for

	/** Finally, add fragment asking if user wants another try. */
	inputMessageErrorString = inputMessageErrorString + userWishFragment;

	/** Return inputMessageErrorString, end method. */
	return inputMessageErrorString;
} // end inputMessageErrorString

/** This method analyses a 2-D array of booleans (width 1). If any of the elements are false, the method returns false. */
/** That is, it will only return true if ALL elements are true. */
public static boolean ifAnyElementIsFalseWholeArrayIsFalse (boolean [] testeeArray, int NUMBER_OF_INTEGERS_FOR_2D_GAME) {
```

```
		/** declarations */
		boolean resultBoolean = true;

		/** code */
		for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
			if (testeeArray[heightCounter] == false) {
				resultBoolean = false;
			} // end if
		} // end for

		/** Return result, end method. */
		return resultBoolean;
	} // end ifAnyElementIsFalseWholeArrayIsFalse

	/** If called, this method displays a dialog based on the error string created in 'inputMessageErrorString()' and asks the user to choose
*/
	/** whether to have another try. The YES/NO integer answer is converted to a boolean, which is then returned. */
	public static boolean wishToRetry2D (String inputMessageErrorString, String userName) {

		/** declarations */
		boolean wishToRetry2DBoolean;
		int wishToRetry2DInteger;
		String invalidInputHeadline = "INVALID INPUT";

		/** code */
		/** Display YES/NO dialog box. */
		wishToRetry2DInteger = JOptionPane.showConfirmDialog(null, inputMessageErrorString, invalidInputHeadline,
JOptionPane.YES_NO_OPTION);

		/** Convert integer response to boolean because it's easier to work with 'true' and 'false' than to remember whether '1' means 'yes'
or 'no'! */
		/** YES = 0, NO = 1. */
		/** Also, I want to work with booleans in the method that calls this one. */
		if (wishToRetry2DInteger == 0) {
			wishToRetry2DBoolean = true;
		}
		else wishToRetry2DBoolean = false;
		// end if

		/** Return result, end method. */
		return wishToRetry2DBoolean;
	} // end wishToRetry2D

	/** This method copies each row of the random and user 2-D arrays into 1-D arrays, then uses 'NumberOfBullseyes()' to populate yet another
```

```
1-D array */
	/** with the number of bulleyes in each row of the user's input. */
	public static int [] getBullseyesArray (int [] [] random2DArray, int [] [] user2DInputArray, int NUMBER_OF_INTEGERS_FOR_2D_GAME, int
INTEGER_LENGTH) {

		/** declarations */
		int [] rowOfRandomArray = new int [INTEGER_LENGTH];
		int [] rowOfUserArray = new int [INTEGER_LENGTH];
		int [] bullseyesArray = new int [NUMBER_OF_INTEGERS_FOR_2D_GAME];

		/** code */
		/** Copy elements of each row of random and user arrays into 1-D arrays. */
		for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
			for (int widthCounter = 0; widthCounter < INTEGER_LENGTH; widthCounter++) {
				rowOfRandomArray[widthCounter] = random2DArray[widthCounter][heightCounter];
				rowOfUserArray[widthCounter] = user2DInputArray[widthCounter][heightCounter];
			} // end width for loop
			bullseyesArray[heightCounter] = numberOfBullseyes(INTEGER_LENGTH, rowOfRandomArray, rowOfUserArray);
		} // end height for loop

		/** Return result, end method. */
		return bullseyesArray;
	} // end getBullseyesArray

	/** This method copies each row of the random and user 2-D arrays into 1-D arrays, then uses 'NumberOfBullseyes()' to populate yet another
1-D array */
	/** with the number of bulleyes in each row of the user's input. */
	/** If only it were possible to pass the 1-D arrays from the above 'getBullseyesArray()' method to this method! */
	public static int [] getHitsArray (int [] [] random2DArray, int [] [] user2DArray, int NUMBER_OF_INTEGERS_FOR_2D_GAME, int
INTEGER_LENGTH) {

		/** declarations */
		int [] rowOfRandomArray = new int [INTEGER_LENGTH];
		int [] rowOfUserArray = new int [INTEGER_LENGTH];
		int [] hitsArray = new int [NUMBER_OF_INTEGERS_FOR_2D_GAME];

		/** code */
		/** Copy elements of each row of random and user arrays into 1-D arrays. */
		for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
			for (int widthCounter = 0; widthCounter < INTEGER_LENGTH; widthCounter++) {
				rowOfRandomArray[widthCounter] = random2DArray[widthCounter][heightCounter];
				rowOfUserArray[widthCounter] = user2DArray[widthCounter][heightCounter];
			} // end width for loop
			hitsArray[heightCounter] = numberOfHits(INTEGER_LENGTH, rowOfRandomArray, rowOfUserArray);
```

```
	} // end height for loop

	/** Return result, end method. */
	return hitsArray;
} // end getHitsArray

/** This method displays the user's score for a single round in the 2-D game, then asks the user if it wants another round. */
/** The user's response is (initially the integer resulting from a YES/NO dialog) is converted to a boolean. */
public static boolean display2DScore(int NUMBER_OF_INTEGERS_FOR_2D_GAME, int [] bullseyesArray, int [] hitsArray,
		int bullseyes, int hits, int INTEGER_LENGTH, int numberOfTries, String userName) {

	int rowCounter;
	String bullseyesFragment;
	String hitsFragment;
	String scoreMessage = "";
	String headline = "YOUR SCORE FOR THIS ROUND";
	int anotherRoundInteger = 1;
	boolean anotherRoundBoolean;

	/** code */
	/** Report scores for each row of current round. */

	/** Construct score message, use it to build dialog asking user if it wants another round. */
	for (int heightCounter = 0; heightCounter < NUMBER_OF_INTEGERS_FOR_2D_GAME; heightCounter++) {
		rowCounter = heightCounter + 1;

		/** Sort singular/plural issues in report. */
		if (bullseyesArray[heightCounter] == 1) {
			bullseyesFragment = " bullseye ";
		}
		else bullseyesFragment = " bullseyes ";
		// end if

		if (hitsArray[heightCounter] == 1) {
			hitsFragment = " hit.";
		}
		else hitsFragment = " hits.";
		// end if

		/** Construct grammatically-correct score message from fragments. */
		scoreMessage = scoreMessage + "In row " + rowCounter + " , you scored " + bullseyesArray[heightCounter] +
				bullseyesFragment + "and " + hitsArray[heightCounter] + hitsFragment + "\n";
	} // end for
```

```
            /** Add on all-important question! */
            scoreMessage = scoreMessage + "Would you like another round?";

            /** Report score and ask the question! Fortunately there is no sign of Noel Edmonds! */
            anotherRoundInteger = JOptionPane.showConfirmDialog(null, scoreMessage, headline, JOptionPane.YES_NO_OPTION);

            /** Convert user's response into a boolean: YES = 0, NO = 1. */
            /** Also, I want to work with booleans in the method that calls this one. */
            if (anotherRoundInteger == 0) {
                  anotherRoundBoolean = true;
            }
            else anotherRoundBoolean = false;
            // end if

            /** Return whether the user wants another round, end method. */
            return anotherRoundBoolean;
      } // end display2DScore

      /** This method is just for for fans of Douglas Adams. */
      public static void terminate() {
            JOptionPane.showMessageDialog(null, "So long and thanks for all the fish.                  ", "The dolphins' last message to
userkind", JOptionPane.INFORMATION_MESSAGE);
      } // end terminate

}
```