```
package CyrilleLingaiJonathanGrant_06;

 * ComputerScience_02_01

import java.io.File;

/**
 * Sort lists of integers by shell sorting and quick sorting algorithms.
 *
 * @author Cyrille Lingai, Grant Jonathan.
 * @version 12/05/19.
 */

public class CyrilleLingaiJonathanGrant_06 {

    // Declaring class variables.

    private final static String inputFileName = "2050 Project 06_Input.txt";
                                    // The file of random integers.
    private final static int sequenceLength = 100;
                                    // The length of the list of integers.

    /**
     * Main execution of program to scan input files, sort lists of integers,
     * and write those lists to independent files.
     *
     * @param       args                    The IO streams for reading and writing files.
     * @throws      FileNotFoundException   If the file is not found.
     * @throws      IOException             If there is not to a writable file.
     */

    public static void main(String[] args) {

        // Declaring local variables.

        int lineNumber = 0;         // Track the line an expression is scanned from.
        int fileNumber = 1;         // The file number to identify and scan files.

        int[] quickSortedArray = new int[sequenceLength];
                                    // The quick sorted integers to write to file.
        int[] shellSortedArray = new int[sequenceLength];
                                    // The shell sorted integers to write to file.

        String outputFileName = "2050 Project 06_OutputX.txt";
                                    // The file of corresponding sorted integers.

        File inputFile = new File(inputFileName);   // The tool for openning files.
        FileWriter fileWriter = null;               // The tool for writing to files.
        Scanner fileScanner = null;                 // The tool to read file.

        // Get input from user and find the corresponding file.

        try {

            fileScanner = new Scanner(inputFile); // Use the reading tool on the file.

            // Scan the input file for all integers and push eac integer to two lists.

            while (fileScanner.hasNextInt()) {
                quickSortedArray[lineNumber] = fileScanner.nextInt(); // Read the line
                shellSortedArray[lineNumber] = quickSortedArray[lineNumber++];
            } // End while.

            fileScanner.close();

        } // End try.
        catch (FileNotFoundException e) {
            System.err.println(e);
        } // End catch.
```

```java

        // Sort both lists of integers.

        shellSort(shellSortedArray);
        quickSort(quickSortedArray, 0, sequenceLength - 1);

        // Attempt to create new files of the specified names,
        // and write the sorted lists to these files.

        try {

            // Write the shell sorted array to file.

            outputFileName = outputFileName.replace("X", Integer.toString(fileNumber));
            fileWriter = new FileWriter(outputFileName);
            fileWriter.write("Shell Sorted Array\n\n");
            fileWriter.write(arrayToString(shellSortedArray));
            fileWriter.close();

            // Open a new file to write the quick sorted array.

            outputFileName = outputFileName.replace(
                    Integer.toString(fileNumber++), Integer.toString(fileNumber));

            // Write the quick sorted array to file.

            fileWriter = new FileWriter(outputFileName);
            fileWriter.write("Quick Sorted Array\n\n");
            fileWriter.write(arrayToString(quickSortedArray));;
            fileWriter.close();

        } // End try.
        catch (IOException e) {
            System.err.println(e.getMessage());
        }  // End catch.

    } // End main method.

// ***************************************************************************

    /**
     * Convert a sorted list of integers to a string with 10 integers per line
     * for writing to file.
     *
     * @param   sortedArray The sorted list of integers.
     */

    private static String arrayToString(int[] sortedArray) {

        // Declaring local variables.

        String arrayString = "";

        // Pushing each integer of the sorted list to a string for the output file.

        for (int i = 0; i < sortedArray.length;) {

            for (int j = 0; j < 10; j++, i++) {
                arrayString += sortedArray[i] + " ";
            } // End for.

            arrayString += "\n";

        } // End for.

        return arrayString;

    } // End arrayToString method.

```

```java
// ****************************************************************************

    /**
     * Sort integers using the shell sorting algorithm.
     *
     * @param   unsortedArray   The list of unsorted integers to sort.
     */

    private static void shellSort(int[] unsortedArray) {

        // Declaring local variables.

        int nextInteger;    // The next integer in the sublist.
        int index = 0;      // The current sublist of integers.

        // Iterate the list at intevals, dividing by 2 each time.

        for (int space = unsortedArray.length/2; space > 0; space /= 2) {

            // Scan the sublist by each interval.

            for (int i = space; i < unsortedArray.length; i++) {

                // Sort the sublist.

                nextInteger = unsortedArray[i];

                for (index = i; index >= space
                        && unsortedArray[index - space] > nextInteger;
                        index -= space) {

                    unsortedArray[index] = unsortedArray[index - space];

                } // End for.

                unsortedArray[index] = nextInteger;

            }   // End for.

        } // End for.

    } // End shellSort method.

// ****************************************************************************

    /**
     * Quicksort implements the textbook case of quicksort to efficiently sort the
     * list by finding a pivot, swapping integers to the correct side of the pivot,
     * splitting each side of the pivot into sublists, and repeating until each
     * sublist has less than four elements. Short sublists of less than four
     * integers are sorted using shell sort: an improved insertion sort.
     *
     * @param   anArray The unsorted list of integers.
     */

    private static void quickSort(int[] anArray, int firstIndex, int lastIndex) {

        // Declaring local variables.

        int pivotIndex = 0;             // The pivot of the sublist.
        final int MIN_SIZE = 4;         // The minimum size of the list.
        int temp = 0;                   // Placeholder to swap two integers.
        int midIndex = (lastIndex - firstIndex) / 2;
                                        // Find the middle integer of the list.
        int leftIndex = firstIndex + 1; // Leftmost integer of the sublist.
        int rightIndex = lastIndex - 2; // Rightmostmost integer of the sublist.
        int pivotValue = 0;             // The integer that seperates two sublists.

        // Sort the sublist using insertion(shell) sort if the length is less than four,
        // otherwise, sort the sublist by the quick sort algorithm.

        if (lastIndex - firstIndex < MIN_SIZE) {

            shellSort(anArray);

        } // End if.
        else {

            // Exchange values of middle and last integers of sublist.

            temp = anArray[midIndex];
            anArray[midIndex] = anArray[lastIndex - 1];
            anArray[lastIndex - 1] = temp;

            // Find the new pivot of sublist.

            pivotIndex = lastIndex - 1;
            pivotValue = anArray[pivotIndex];

            // Scan the sublist and swap integers.

            while (anArray[leftIndex] < pivotValue) {
                leftIndex++;
            } // End while.

            while (anArray[rightIndex] > pivotValue) {
                rightIndex++;
            } // End while.

            if (leftIndex < rightIndex) {

                temp = anArray[leftIndex];
                anArray[leftIndex] = anArray[rightIndex];
                anArray[rightIndex] = temp;

            } // End if.

            // Swap the pivot and leftmost integer of the sublist.

            temp = anArray[pivotIndex];
            anArray[pivotIndex] = anArray[leftIndex];
            anArray[leftIndex] = temp;
            pivotIndex = leftIndex;

            // Recursively quick sort the new sublists.

            quickSort(anArray, firstIndex, pivotIndex - 1);
            quickSort(anArray, pivotIndex + 1, lastIndex);

        } // End else.

    } // End quickSort method.

} // End class.
```