1 Background

For Project 1, the students were asked in §5 *Software Design*, item 1 of the project document to put all of their code in a class named *Main*, so they should have submitted a source code file named *Main.java*. To help the students, I provided a somewhat detailed pseudocode design in §6 of the project document. I did not require the students to implement this design but I would expect that the majority of them did. If so, then the template for the *Main* class would be:

```
/**
 * Header comment block. The required contents is specified in the project document in §5 Item 10.
 */

// I have shown the students how to create a package and place their class declaration inside the package.
// In this document we will assume the student named the package p01. The student may not have declared a
// package and may have just placed their class declarations in the default unnamed package, which is fine.
package p01;

public class Main {
    // Declare static final constants for RUNS_UP and RUNS_DOWN

    // Note: The project document required that Main does not declare any instance variables, so while
    // grading, we will examine their class for such declarations and will deduct a small penalty of -1 pt
    // if they did.

    // main() should instantiate a Main object and then call run() on the object. Other than main(), there
    // shall not be any static/class methods.
    public static void main(String[] pArgs) {
    }

    // This ctor is empty. If the student omitted it, we will not penalize them.
    public Main() {
    }

    public void run() {
    }

    // This method is the solution to Homework Assignment 1, Exercise 3.5.
    private ArrayList<Integer> arrayListCreate(int pLen, int pInitValue) {
    }

    private ArrayList<Integer> findRuns(ArrayList<Integer> pList, int pDir) {
    }

    private ArrayList<Integer> mergeLists(ArrayList<Integer> pListRunsUp, ArrayList<Integer> pListRunsDn) {
    }

    private ArrayList<Integer> readInputFile(String pFname) throws FileNotFoundException {
    }

    private void writeOutputFile(String pFilename, ArrayList<Integer> pListRuns)
    throws FileNotFoundException {
    }
}
```

Note that *readInputFile*() and *writeOutputFile*() should throw a *FileNotFoundException* when the input or output file cannot be opened for reading or writing. In this case, *run*()—which calls these two methods—will catch the exceptions. In the exception handlers, the program should output an error message stating that the input file *p01-in.txt* could not be opened for reading or the output file *p01-runs.txt* could not be opened for writing. Then, the program should terminate itself by calling the *System.exit*() method. See Software Requirements 5 and 6 in the Project 1 document. Alternatively, the student may put the try/catch blocks in *readInputFile*() and *writeOutputFile*().

2 The Grading Archive – *cse205-p01-grader.zip*

When you extract the grading archive, you will see this directory structure shown in Fig. 1 below. The *subs* directory contains the student's submitted zip archives named as *asuriteid*-p01.zip. We will discuss the files in testcases in a bit.

cse205-p01-grader	Grading rubric archive top-level directory
doc	Documents
cse205-p01.pdf	The project document given to the students
cse205-p01-rubric.pdf	The solution & grading rubric document not given to the students
roster	Gradebook file
roster.csv	Download gradebook from Canvas in CSV format
src	Source code directory
Main.class	The compiled Main.java
Main.java	My solution code in a class named Main
subs	Contains submission related files
asuriteid.zip	The submitted solution zip archive for each student
submissions	Zip archive download from Canvas containing all of the solutions
testcases	Testing files
testcase1	Input and correct output files for Test Case 1
correct.txt	The output from the instructor's program
p01-in.txt	Test Case 1 input file
testcase2	Input and correct output files for Test Case 2
correct.txt	The output from the instructor's program
p01-in.txt	Test Case 2 input file
...	
testcase7	Input and correct output files for Test Case 7
correct.txt	The output from the instructor's program
p01-in.txt	Test Case 7 input file

Figure 1: The contents of the grading archive

3 Building and Running when the Student Declared Packages

As mentioned in §1, I have shown the students how to create a package containing their class files by writing a line such as this near the top of their source code file:

```
package p01;
```

When you build and then run their program, you may see an error message similar to the one below, where in this case, *Main* is the name of the class containing the *main*() method:

```
Error: Main method not found in class Main, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Or, you may see this error message:

```
Error: Could not find or load main class Main
```

If the student did not create a package, then they placed their class declaration in the *default unnamed package*; in this case, when running, you will not see these errors. They will only occur when the student created a package. They occur on our end, are not the fault of the student, and we will not penalize them for properly using packages—as that is what I want them to do.

There are two ways we can work around the problem: the proper way and the easy way. Since the proper way is slightly more difficult and time consuming than the easy way, then we will use the easy way. The easy way is to simply edit their source code file and either comment out or delete the `package` line near the top of the file. Then, rebuild and the program should run fine.

4 Performing Test Cases 1–7

The *cse205-p01-grader/testcases* directory contains seven subdirectories. Each testcase subdirectory contains two files: (1) *p01-in.txt* is the input file the student's program will open and read during testing. This file contains the list of integers for which we are determining the runs up and runs down counts; and (2) *correct.txt* is the correct output from the instructor's program given the corresponding *p01-in.txt* input file.

When grading a student's project, start by copying *testcase1/*.txt* to the working directory. Run the student's program and it should open the testing input file named *p01-in.txt* and produce the output file named *p01-runs.txt*. Compare the correct output from the instructor's program in *correct.txt* to the student's output in *p01-runs.txt*. Before considering that the test case failed, look for minor formatting issues. For example, the output should be something like this:

```
runs_total: 7
runs_1: 4
runs_2: 2
runs_3: 0
...
```

Suppose that rather than outputting a colon, the program outputted a comma:

```
runs_total, 7
runs_1, 4
runs_2, 2
runs_3, 0
...
```

If that is the only difference, i.e., the numbers must match up, then we do not want to penalize the student for such a minor mistake, so assume in this case that the test case passed. Or, suppose that instead of outputting underscores between "runs" and the value of k, the program outputted spaces:

```
runs total: 7
runs 1: 4
runs 2: 2
runs 3: 0
...
```

Again, we will not penalize them for such a minor mistake. Or, if the student just omitted the colon altogether, then do mark off for this. Another mistake that we may see is that the student's program may not output the final $runs_k$ value. For example, suppose the length of the input list is $n = 5$. Then k should be $k = 1, 2, 3, 4$. Suppose the output is:

```
runs_total: 7
runs_1: 4
runs_2: 2
runs_3: 0
```

rather than:

```
runs_total: 7
runs_1: 4
runs_2: 2
runs_3: 0
runs_4: 0
```

This happens due to a bug in the **for** loop which prints out the $runs_k$ values. This bug is slightly more than a minor mistake but it is not so serious that we would want to subtract an entire point for it. In this case, as long as the numbers match up, then only deduct **-0.5 pts** (a 2% deduction) **and only apply this deduction once**, i.e., in general, **we do**

not penalize a student multiple times for the same coding mistake. Some students may print the string literal "runs_k" rather than "runs_" concatenated with the integer value of k

```
runs_total: 7
runs_k: 4
runs_k: 2
runs_k: 0
```

rather than this:

```
runs_total: 7
runs_1: 4
runs_2: 2
runs_3: 0
```

For this we deduct only **0.5 pt** (a 2% deduction) and only apply this deduction once. As you are grading, you may see situations which the instructor did not think of, so if you are unsure if you should consider that the test case passed or failed, then contact the instructor. Ultimately, however, if the numbers do not match, even if it is only one number, then the test case fails and the penalty for a failed test case is **-0.5 pt** (a 2% deduction per each).

5 Performing Test Cases 8 and 9

The next pair of test cases will test that the program properly handles the *FileNotFoundException* (FNFE) which gets thrown when the input and output files cannot be opened.

Test case 8 is to verify the student's program catches and handles the FNFE that gets thrown when the *p01-in.txt* input file is missing and cannot be opened. To do this, delete all of the *.txt* files from the working directory and then run the program. It should output a message similar to that shown in the project document in §4 Item 5. I did not dictate in the project document what the displayed error message should be. As long as the message includes the phrases **input file** and/or **for reading**, then that is sufficient. After displaying the message, the program should terminate by calling the *System.exit*() method. If the program fails this test case, then the penalty deduction is **-0.5 pt**.

The final testcase, test case 9, is to verify the student's program catches and handles the FNFE that gets thrown when the *p01-runs.txt* output file cannot be opened for writing. The easiest way to make this happen is to clear the file *write* permission bit to make the file *read-only*. To do this in Bash, we will create the file *p01-runs.txt* in the student's directory and then we will use the `chmod` command to clear the *write* bit on *p01-runs.txt*, and then run the program. This can be done in Bash using these commands:

```
$ cat <<< LISA > p01-runs.txt
$ chmod -w p01-runs.txt
$ java Main
```

Again, I did not dictate what the error message should say, as long as it includes the phrases **output file** and/or **for writing**. If the program fails this test case, then deduct **-0.5 pt**. After this test, to make the output file writable again, use the command: `chmod +w p01-runs.txt`.

Note that the input FNFE and the output FNFE handlers must display **different messages** (see the project document §5 Items 4-5). If the displayed message is the same for both types of FNFE's, then deduct **-0.5 pt** for the incorrect message but do not deduct this point if you already deducted -1 pt because the test case failed (i.e., the FNFE was not even caught).

6 Did not Follow the Submission Instructions

Some students do not follow the submission instructions. This is inconvenient because it will cause a grading script to fail when files which are supposed to be named *abc* are instead named *xyz*, and similarly for directories, class names, method names, and so on. These cases are handled in Step 2 of the grading rubric.

1. If the student submitted a ZIP archive but did not place *Main.java* or the submitted *.java* file in a folder named *asuriteid*-p01, then go ahead and build, run, and test the program. The penalty deduction for this case is **-1 pt**.

2. If the student did not submit a source code file, e.g,. perhaps they submitted a PDF or .DOCX file, then we cannot build the program no matter how perfect it may appear. In this case, we will treat the submission as a program which does not compile due to syntax errors, so Step 6 in the grading rubric will apply.

3. In §5, Item 1 of the project document, students were instructed to name their main class *Main*. If the student submitted a Java file which was not named *Main.java* then go ahead and build, run, and test the program. The penalty deduction for this case is **-1 pt**.

Those are all the cases I can think of. If you find a submission which is incorrect and does not seem to match any of these cases, then contact the instructor.

7 Grading Rubric

1. **Start.** Set *score* = 25. If at any time during the grading steps, *score* reaches 0 go to the final step.
2. **Submission Instructions.** See §6 for how to handle submissions where the student did not follow the submission instructions. Set ***score*** **-=** ***penalty*** as required.
3. **Header Comment Block.** The source code file is to contain a header comment block containing the *class name*; a *description* of the contents of the file; course info (*year* and *semester*); the project number (*1*); and the author information (*name*, *asurite id*, and *email address*). If their source code file is missing the header comment block or if most of the required information is missing, then **set** ***score*** **-= 0.5**.
4. **Programming Style.** Examine *Main.java* and if the student's code exhibits poor programming style (primarily we are looking only for proper and consistent indentation from the left margin), then **set** ***score*** **-= 1**.
5. **Object-Oriented Programming.** The project requirement stated that *main*() was to instantiate a *Main* object and then call *run*() on the object. Consequently, other than *main*(), there should be no static/class methods. Examine *Main.java* and if any non-*main*() methods are declared as `static`, then **set** ***score*** **-= 0.5**. Only apply this penalty once.
6. **Using Multiple Methods**. In §5 *Software Design*, Item 5 specifies that the students were to write the code using multiple methods, i.e., to not put all of the code in *main*(). Examine the code and if you see that all of the code is in *main*() or one other function that *main*() calls, e.g., *run*(), then **set** ***score*** **-= 0.5**.
7. **Not Declaring Instance Data**. In §5 Software Design, Item 6 specifies that the students were to not declare any instance variables, i.e, all of the variables should be declared as local variables. Examine the code and if you see that *Main* declares instance data, then **set** ***score*** **-= 0.5** (class constants are fine).
8. **Using Primitive Arrays.** In the project document, §5 *Software Design*, Item 2 specifies that the students were to use the *ArrayList* class and not primitive arrays. Examine the code and if you see that the code uses primitive arrays, then **set** ***score*** **-= 0.5**.
9. **Build.** Note that some of the students may have placed the *Main* class in a package. If so, near the top of *Main.java* will be this line of code: `package` ***`package-name`***`;` This package line may cause the build to fail or the program to not run, so before building, delete the `package` line and then build the project. If the build fails due to syntax errors go to Step 10, otherwise, if the build succeeds go to Step 11.
10. **Build Failed Due to Syntax Errors.** Since the student's project does not compile, we will award partial credit based on how much of the code was completed. See the solution source code in **Appendix A**. Examine the student's source code and determine how much the student completed in comparison to the solution. If it appears that the student made a **reasonably good attempt** to complete the **majority** of the requested code, then **set** ***score*** **-= 8** (this

is a 32% project deduction or +17 pts for a good attempt that does not compile; I do not assign letter grades to projects, but 17/25 is 68% which would be a C). Otherwise, if the code appears to be **substantially incomplete or very wrong**, then **set *score* -= 10** (this is a 40% project deduction or +15 pts for a poor attempt that does not compile; 15/25 is 60% which would be a D+). Go to Step 12.

11. **Build Succeeds, Perform Testing.** The grading zip archive contains seven test cases in subdirectories under the *testing* directory. See §4–§5 for a discussion of the nine test cases and how to perform them. The test cases are:

    Test Case 1: *p01-in.txt* contains [1, 2] (1 run of $k = 1$).
    Test Case 2: *p01-in.txt* contains 1000 random integers (total runs is 664, with $k = 6$ being the longest run).
    Test Case 3: *p01-in.txt* contains 100 integers: [1, 2, 3, ..., 100] (1 run up with $k = 99$).
    Test Case 4: *p01-in.txt* contains 100 integers: [100, 99, 98, ..., 1] (1 run down with $k = 99$).
    Test Case 5: *p01-in.txt* contains 100 integers: [1, 2, 1, 2, 1, 2, ..., 1, 2] (99 runs of $k = 1$).
    Test Case 6: *p01-in.txt* contains 100 integers: [1, 1, 1, ..., 1] (2 runs of $k = 99$).
    Test Case 7: *p01-in.txt* contains 100 integers: [1, 2, 3, 4, 5, ..., 50, 49, 48, ..., 1] (2 runs of $k = 49$).
    Test Case 8: Tests that the program fails gracefully when the input file *p01-in.txt* cannot be opened for reading.
    Test Case 9: Tests that the program fails gracefully when the output file *p01-runs.txt* cannot be opened for writing.

    For each test case failure, **set *score* -= 0.5**. The student can lose **a maximum of 4.5 pts during testing** so their maximum project score would be 25 - 4.5 = 20.5 pts (which would be 82% or a B). Note: If the program crashes with an exception other than *FileNotFoundException*, or gets stuck in an infinite loop during testing, then treat this as a test case failure. Go to Step 12.

12. **Done Grading.** Enter *round*(*score*) in the Canvas gradebook. **This is very important:** You must make remarks in the gradebook **Comment** box explaining where, why, and how many points were deducted. Then, please **write your initials** in the **Comment** box so the student will know which grader graded their project.