

<Student Name>

SOFTWARE LIFE CYCLE REPORT – FOR LAB ASSIGNMENT <number>

PHASE 1: SPECIFICATION (“What do we build?”)

Write a class **GradeBook** that can:

- a) Create objects (grade books) with an arbitrary course name and an arbitrary array of grades.
- b) Find the lowest, highest, and average grades.
- c) Display the grade distribution as a bar graph like the following:

```
00-09:
10-19:
20-29:
30-39:
40-49:*
50-59:
60-69:*
70-79:*****
80-89:*****
90-99:***
100:*
```

Test the class by performing the following steps:

- a) Create a new **GradeBook** object with the course name: “CS1110 Computer Science I” and grades: 76, 88, 89, 82, 96, 100, 84, 77, 75, 32, 85, 62, 100, 92, 85
- b) Display the course name, lowest, highest, and average grades.
- c) Display the grade distribution as a bar graph.
- d) Create another new **GradeBook** object with the course name: “CS1120 Computer Science II” and grades: 68, 88, 98, 29, 61, 100, 87, 77, 53, 28, 56, 62, 95, 85, 75, 76
- e) Display the new course name, lowest, highest, and average grades.
- f) Display the new grade distribution as a bar graph.

PHASE 2: DESIGN

2.1 Modules and Their Basic Structure

My program will have two modules.

- a) Module 1: Class **GradeBook** will contain:
 - i) a private field (and public accessor/getter and mutator/setter **getX/setX** methods for it if needed) representing the name of the course, and a private array representing the grades
 - ii) a constructor with two parameters: the course name and the grades.
 - iii) a method **findMinimum** that finds the lowest grade
 - iv) a method **findMaximum** that finds the highest grade
 - v) a method **computeAverage** that finds the average grade
 - vi) a method **displayDistribution** that displays the grade distribution in bar graph form.
- b) Module 2: The testing module **GradeBookTest** will contain:
 - i) The method **main** that performs all tasks (specified above and in Section 2.2 below) required to test **GradeBook** according to the specs.

2.2 Pseudocode for the Modules

2.2.1 Pseudocode for GradeBook

1a) GradeBook Pseudocode Refinement #1:

Notes for students:

- 1) Pseudocode is developed in the form of Java comments. Thanks to this, the final Pseudocode Refinement can be developed using NetBeans, or be copied into NetBeans after developing it using other tools (e.g., an editor).
- 2) Your report does not need to include strikethroughs, highlighting, or textboxes. This is to show you the process.

```
// Fields: courseName, grades (array)

// Constructor: Initialize fields courseName and grades (with
// values from parameters).

// Methods: -----

// Method findMinimum: Find the minimum value in grades.

// Method findMaximum: Find the maximum value in grades.

// Method computeAverage: Compute the average of all the values in grades.

// Method displayDistribution: Display the frequency of the grades (as
// required by in the Specification phase).
```

1b) GradeBook Pseudocode Refinement #2:

Notes for students:

- 1) At each pseudocode refinement step, some lines of pseudocode represent a sequence of steps. We add beneath them more detailed pseudocode.
- 2) To indicate better what has changed in developing Pseudocode Refinement #N into Pseudocode Refinement #(N+1), new refinements of the pseudocode are highlighted.
- 3) Notice indentation of newly added pseudocode w.r.t. the “old” pseudocode. This is essential for proper structuring and orderly logic outline.
- 4) At this step, the “old” pseudocode represents the behavior of the method or constructor as a whole, while the “new” pseudocode represents the refined steps within, and so it is indented.

```
// Fields: courseName, grades (array)

// Constructor: Initialize fields courseName and grades (with
// values from parameters).
// Set courseName
// Set grades

// Methods: -----

// Method findMinimum: Find the minimum value in grades.
// Initialize lowest grade as first element in grades
// For each grade do:
//     If grade is lower than lowest grade, assign it to lowest grade
// Return lowest grade

// Method findMaximum: Find the maximum value in grades.
// Initialize highest grade as first element in grades
```

```

    // For each grade do:
    //     If grade is higher than highest grade, assign it to highest grade
    // Return highest grade

// Method computeAverage: Compute the average of all the values in grades.
// Initialize total to 0
// For each grade do:
//     Add grade to total
// Compute and return average by dividing total by number of grades

// Method displayDistribution: Display the frequency of the grades (as
// required by in the Specification phase).
// Display header
// Initialize an array to hold frequency of grades in each grade range
// For each grade do:
//     Increment the appropriate frequency
// For i = 0 to frequency length - 1 do:
//     Print bar in chart for frequency[i]

```

1c) GradeBook Pseudocode Refinement #3:

Notes for students:

- 1) In this refinement, only one line of pseudocode still represents a sequence of steps. We add beneath it more detailed pseudocode.
- 2) Notice again indentation of newly added pseudocode w.r.t. the “old” pseudocode.

```

// Fields: courseName, grades (array)

// Constructor: Initialize fields courseName and grades (with
// values from parameters).
// Set courseName
// Set grades

// Methods: -----

// Method findMinimum: Find the minimum value in grades.
// Initialize lowest grade as first element in grades
// For each grade do:
//     If grade is lower than lowest grade, assign it to lowest grade
// Return lowest grade

// Method findMaximum: Find the maximum value in grades.
// Initialize highest grade as first element in grades
// For each grade do:
//     If grade is higher than highest grade, assign it to highest grade
// Return highest grade

// Method computeAverage: Compute the average of all the values in grades.
// Initialize total to 0
// For each grade do:
//     Add grade to total
// Compute and return average by dividing total by number of grades

// Method displayDistribution: Display the frequency of the grades (as
// required by in the Specification phase).
// Display header
// Initialize an array to hold frequency of grades in each grade range
// For each grade do:
//     Increment the appropriate frequency
// For i = 0 to frequency length - 1 do:
//     Print bar in chart for frequency[i]
// If i == 10 (i.e. last grade range: 100) then:

```

Only this line of pseudocode is further refined.

```
// Print range label " 100: "
// Else:
// Print range label ("[i * 10]-[i * 10 + 9]: ")
// For numStars = 0 to frequency[i] do
// Print "*"
// Print new line
```

2.2.2. Pseudocode for GradeBookTest

2a) Pseudocode Refinement #1:

```
// Create a new GradeBook object for first set of data.
// Display the course name, lowest, highest, and average grades.
// Display the grade distribution as a bar graph (using method from GradeBook).

// Create another new GradeBook object for second set of data.
// Display the new course name, lowest, highest, and average grades.
// Display the new grade distribution as a bar graph (using method from
// GradeBook)
```

Note for students:

- 1) Pseudocode Refinement #1 for GradeBookTest is sufficiently detailed. No Pseudocode Refinement #2 is needed.

PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)

No risks (to timetable, cost, human health, etc.) are identified by me.

PHASE 4: VERIFICATION (“Are the algorithms correct?”)

The algorithm has only one execution path (a sequential execution). Correctness of the path has been verified by me by analyzing its steps, and their completeness w.r.t. the Specification.

PHASE 5: CODING

5a) Code Refinement #1 (class structure with pseudocode only; pseudocode is used as comments)

Notes for students:

- 1) This *Code* Refinement defines the “external” class structure (without defining the “internal” class structure, that is methods, etc., within classes) as developed in the Design phase (Section 2.1).
- 2) All pseudocode developed in the Design phase (Section 2.2) is copied into the appropriate classes in this Code Refinement #1, and it appears in them as comments.
- 3) New elements added in Step 5a are highlighted (the rest is the most refined pseudocode copied from Section 2.2—more precisely, from 1c and 2a from this section.).

File GradeBook.java:

```
package sample1c;

// The GradeBook class.
public class GradeBook
{
    // Fields: courseName, grades (array)

    // Constructor: Initialize fields courseName and grades (with
    // values from parameters).
```

```

        // Set courseName
        // Set grades

// Methods: -----
// Method findMinimum: Find the minimum value in grades.
// Initialize lowest grade as first element in grades
// For each grade do:
//     If grade is lower than lowest grade, assign it to lowest grade
// Return lowest grade

// Method findMaximum: Find the maximum value in grades.
// Initialize highest grade as first element in grades
// For each grade do:
//     If grade is higher than highest grade, assign it to highest grade
// Return highest grade

// Method computeAverage: Compute the average of all the values in grades.
// Initialize total to 0
// For each grade do:
//     Add grade to total
// Compute and return average by dividing total by number of grades

// Method displayDistribution: Display the frequency of the grades (as
// required by in the specification phase).
// Display header
// Initialize an array to hold frequency of grades in each grade range
// For each grade do:
//     Increment the appropriate frequency
// For i = 0 to frequency length - 1 do:
//     Print bar in chart for frequency[i]
//         If i == 10 (i.e. last grade range: 100) then:
//             Print range label " 100: "
//         Else:
//             Print range label ("[" + i * 10 + "]-[" + i * 10 + 9 + "]: ")
//         For numStars = 0 to frequency[i] do
//             Print "*"
//         Print new line
} // end class GradeBook

```

File GradeBookTest.java:

```

package samples1c;

// The GradeBookTest class for running (thus testing) the GradeBook class.
public class GradeBookTest
{
    // Create a new GradeBook object for first set of data.
    // Display the course name, lowest, highest, and average grades.
    // Display the grade distribution as a bar graph (using method from GradeBook).

    // Create another new GradeBook object for second set of data.
    // Display the new course name, lowest, highest, and average grades.
    // Display the new grade distribution as a bar graph (using method from
    //     GradeBook)
} // end class GradeBookTest

```

5b) Code Refinement #2 (still incomplete program: class and constructor/method structure with pseudocode only; pseudocode is used as comments)

Notes for students:

- 1) This *Code Refinement* defines the “internal” class structure (constructors, methods) as developed in the Design phase in Section 2.1.

2) New elements added in Step 5b are highlighted.

File GradeBook.java:

```
package sample1c;

// The GradeBook class.
public class GradeBook
{
    // Fields: courseName, grades (array)

    // Constructor: Initialize fields courseName and grades (with
    // values from parameters).
    public GradeBook(/* name, gradesArray */) {
        // Set courseName
        // Set grades
    } // end constructor

    // Methods: -----

    // Method findMinimum: Find the minimum value in grades.
    public int findMinimum() {
        // Initialize lowest grade as first element in grades
        // For each grade do:
        //     If grade is lower than lowest grade, assign it to lowest grade
        // Return lowest grade
    } // end method GetMinimum

    // Method findMaximum: Find the maximum value in grades.
    public int findMaximum() {
        // Initialize highest grade as first element in grades
        // For each grade do:
        //     If grade is higher than highest grade, assign it to highest grade
        // Return highest grade
    } // end method findMaximum

    // Method computeAverage: Compute the average of all the values in grades.
    public double computeAverage() {
        // Initialize total to 0
        // For each grade do:
        //     Add grade to total
        // Compute and return average by dividing total by number of grades
    } // end method computeAverage

    // Method DisplayDistribution: Display the frequency of the grades (as
    // required by in the Specification phase).
    public void displayDistribution() {
        // Display header
        // Initialize an array to hold frequency of grades in each grade range
        // For each grade do:
        //     Increment the appropriate frequency
        // For i = 0 to frequency length - 1 do:
        //     Print bar in chart for frequency[i]
        //         If i == 10 (i.e. last grade range: 100) then:
        //             Print range label " 100: "
        //         Else:
        //             Print range label ("[i * 10]-[i * 10 + 9]: ")
        //         For numStars = 0 to frequency[i] do
        //             Print "*"
        //         Print new line
    } // end method displayDistribution
} // end class GradeBook
```

File GradeBookTest.Java:

```
package sample1c;

// The GradeBookTest class for running (thus testing) the GradeBook class.
public class GradeBookTest
{
    // Main method begins program execution
    static void main(String[] args) {
        // Create a new GradeBook object for first set of data.
        // Display the course name, lowest, highest, and average grades.
        // Display the grade distribution as a bar graph (using method from GradeBook).

        // Create another new GradeBook object for second set of data.
        // Display the new course name, lowest, highest, and average grades.
        // Display the new grade distribution as a bar graph (using method from
        // GradeBook)
    } // end Main
} // end class GradeBookTest
```

5c) Code Refinement #3 (complete program--with complete fields/properties, code for constructor/methods)

Notes for students:

- 1) At this point, we have class and class member structure defined, imposing the proper structure over the fully refined pseudocode.
- 2) In general, code should be added directly below the most refined pseudocode level that it implements. This is what you see in most cases below.
- 3) In some cases, pseudocode is so detailed, that the code is very similar to it. In these cases, the code can *replace* the pseudocode, to improve readability (see the boxed comments indicating such cases.) When pseudocode is replaced with code, clarifying information should be retained as comments (see clearly indicated examples below, e.g., for “If i for the last grade range: 100” close to the end of the code for the displayDistribution method).
- 4) The code immediately below includes many comments for students. To help you in deciding what should appear in your SLC reports, a cleaned up version of the code follows the code with extra comments for students.

File GradeBook.Java:

```
package sample1c;

// The GradeBook class.
public class GradeBook
{
    // Fields: courseName, grades (array)
    private String courseName;
    private int[] grades;

    // getter for courseName
    public String getCourseName() {
        return courseName;
    }

    // Constructor: Initialize fields courseName and grades (with
    // values from parameters).
    public GradeBook(String name, int[] gradesArray) {
        // Set courseName
        courseName = name;
        // Set grades
        grades = gradesArray;
    } // end constructor

    // Methods: -----
```

Code that is very similar to pseudocode can simply *replace* the pseudocode. Here are 3 examples (over 10 more cases or such replacement of pseudocode by code are below, without arrows pointing to the replacing code).

```

// Method findMinimum: Find the minimum value in grades.
public int findMinimum() {
    // Initialize lowest grade as first element in grades
    int lowest = grades[0];
    // For each grade do:
    for (int grade : grades) {
        // If grade is lower than lowest grade, assign it to lowest grade
        if (grade < lowest)
            lowest = grade;
    }
    // Return lowest grade
    return lowest;
} // end method findMinimum

// Method findMaximum: Find the maximum value in grades.
public int findMaximum() {
    // Initialize highest grade as first element in grades
    int highest = grades[0];
    // For each grade do:
    for (int grade : grades) {
        // If grade is higher than highest grade, assign it to highest grade
        if (grade > highest)
            highest = grade;
    }
    // Return highest grade
    return highest;
} // end method findMaximum

// Method computeAverage: Compute the average of all the values in grades.
public double computeAverage() {
    // Initialize total to 0
    int total = 0;
    // For each grade do:
    for (int grade : grades) {
        // Add grade to total
        total += grade;
    }

    // Compute and return average by dividing total by number of grades
    return (double)total / grades.length;
} // end method computeAverage


// Method DisplayDistribution: Display the frequency of the grades (as
// required by in the specification phase).
public void displayDistribution() {
    // Display header
    System.out.println("Grade Distribution");
    // Initialize an array to hold frequency of grades in each grade range
    int[] frequency = new int[11];
    // For each grade do:
    for (int grade : grades) {
        // Increment the appropriate frequency
        frequency[grade / 10]++;
    }

    // Print chart with label and bar for each frequency
    // For i = 0 to frequency length - 1 do:
    for (int i = 0; i < frequency.length; i++) {
        // Print bar in chart for frequency[i]
        // If i == 10 (i.e. last grade range: 100) then:
        if (i == 10) // If i for the last grade range: 100
            // Print range label "100: "
            System.out.print(" 100: ");
        // Else:
        else // else i for any other grade range: e.g. 30-39

```

In the vast majority of cases (like here), pseudocode helps explain the purpose of code, so it should be retained as a comment.

Redundant comments are replaced with code, but clarifying information (such as "If i for the last grade range: 100") should be retained.



```

// Print range label ("[i * 10] - [i * 10 + 9] : ")
    System.out.printf("%02d-%02d: ", i * 10, i * 10 + 9);
// For numStars = 0 to frequency[i] do
    for (int numStars = 0; numStars < frequency[i]; numStars++)
// Print "*"
        System.out.print("*");
// Print new line
    System.out.println();
}
} // end method displayDistribution
} // end class GradeBook

```

File GradeBookTest.Java:

```

package sample1c;

// The GradeBookTest class for running (thus testing) the GradeBook class.
public class GradeBookTest
{
    // Main method begins program execution
    static void main(String[] args) {
        // Create a new GradeBook object for first set of data.
        int[] gradesArray1 =
            { 76, 88, 89, 82, 96, 100, 84, 77, 75, 32, 85, 62, 100, 92, 85 };
        GradeBook gb1 = new GradeBook("CS1110 Computer Science I\n", gradesArray1);
        // Display the course name, lowest, highest, and average grades.
        System.out.println(gb1.getCourseName());
        System.out.printf("Minimum: %s\n", gb1.findMinimum());
        System.out.printf("Maximum: %s\n", gb1.findMaximum());
        System.out.printf("Average: %s\n\n", gb1.computeAverage());
        // Display the grade distribution as a bar graph.
        gb1.displayDistribution();

        System.out.println("\n===== \n");
        // Create another new GradeBook object for second set of data.
        int[] gradesArray2 =
            { 68, 88, 98, 29, 61, 100, 87, 77, 53, 28, 56, 62, 95, 85, 75, 76 };
        GradeBook gb2 = new GradeBook("CS1120 Computer Science II\n", gradesArray2);
        // Display the new course name, lowest, highest, and average grades.
        System.out.println(gb2.getCourseName());
        System.out.printf("Minimum: %s\n", gb2.findMinimum());
        System.out.printf("Maximum: %s\n", gb2.findMaximum());
        System.out.printf("Average: %s\n\n", gb2.computeAverage());
        // Display the new grade distribution as a bar graph.
        gb2.displayDistribution();
    } // end Main
} // end class GradeBookTest

```

What would appear in your SLC reports (the cleaned up version of the above code).

File: GradeBook.java

```

package sample1c;

// The GradeBook class.
public class GradeBook
{
    // Fields: courseName, grades (array)
    private String courseName;
    private int[] grades;

    // getter for courseName

```

```

public String getCourseName() {
    return courseName;
}

// Constructor: Initialize fields courseName and grades (with
// values from parameters).
public GradeBook(String name, int[] gradesArray) {
    courseName = name;
    grades = gradesArray;
} // end constructor

// Methods: -----

// Method findMinimum: Find the minimum value in grades.
public int findMinimum() {
    // Initialize lowest grade as first element in grades
    int lowest = grades[0];

    for (int grade : grades) {
        // If grade is lower than lowest grade, assign it to lowest grade
        if (grade < lowest)
            lowest = grade;
    }

    return lowest;
} // end method findMinimum

// Method findMaximum: Find the maximum value in grades.
public int findMaximum() {
    // Initialize highest grade as first element in grades
    int highest = grades[0];

    for (int grade : grades) {
        // If grade is higher than highest grade, assign it to highest grade
        if (grade > highest)
            highest = grade;
    }

    return highest;
} // end method findMaximum

// Method computeAverage: Compute the average of all the values in grades.
public double computeAverage() {
    // Initialize total to 0
    int total = 0;

    for (int grade : grades) {
        // Add grade to total
        total += grade;
    }

    // Compute and return average by dividing total by number of grades
    return (double)total / grades.length;
} // end method computeAverage

// Method DisplayDistribution: Display the frequency of the grades (as
// required by in the Specification phase).
public void displayDistribution() {
    // Display header
    System.out.println("Grade Distribution");
    // Initialize an array to hold frequency of grades in each grade range
    int[] frequency = new int[11];

    for (int grade : grades) {
        // Increment the appropriate frequency

```

```

        frequency[grade / 10]++;
    }

    // Print chart with label and bar for each frequency
    for (int i = 0; i < frequency.length; i++) {
        if (i == 10) // If i for the last grade range: 100
            system.out.print(" 100: ");
        else // else i for any other grade range: e.g. 30-39
            system.out.printf("%02d-%02d: ", i * 10, i * 10 + 9);
        for (int numStars = 0; numStars < frequency[i]; numStars++)
            system.out.print("*");
        system.out.println();
    }
} // end method displayDistribution
} // end class GradeBook

```

File: GradeBookTest.java

```

package samples1c;

// The GradeBookTest class for running (thus testing) the GradeBook class.
public class GradeBookTest {
    // Main method begins program execution
    static void Main(String[] args) {
        // Create a new GradeBook object for first set of data.
        int[] gradesArray1 =
            { 76, 88, 89, 82, 96, 100, 84, 77, 75, 32, 85, 62, 100, 92, 85 };
        GradeBook gb1 = new GradeBook("CS1110 Computer Science I\n", gradesArray1);
        // Display the course name, lowest, highest, and average grades.
        System.out.println(gb1.getCourseName());
        System.out.printf("Minimum: %s\n", gb1.findMinimum());
        System.out.printf("Maximum: %s\n", gb1.findMaximum());
        System.out.printf("Average: %s\n\n", gb1.computeAverage());
        // Display the grade distribution as a bar graph.
        gb1.displayDistribution();

        System.out.println("\n=====");
        // Create another new GradeBook object for second set of data.
        int[] gradesArray2 =
            { 68, 88, 98, 29, 61, 100, 87, 77, 53, 28, 56, 62, 95, 85, 75, 76 };
        GradeBook gb2 = new GradeBook("CS1120 Computer Science II\n", gradesArray2);
        // Display the new course name, lowest, highest, and average grades.
        System.out.println(gb2.getCourseName());
        System.out.printf("Minimum: %s\n", gb2.findMinimum());
        System.out.printf("Maximum: %s\n", gb2.findMaximum());
        System.out.printf("Average: %s\n\n", gb2.computeAverage());
        // Display the new grade distribution as a bar graph.
        gb2.displayDistribution();
    } // end Main
} // end class GradeBookTest

```

PHASE 6: TESTING ("Did we build it correctly?")

This program has only a single execution path. Therefore a single run tests it completely. The single test produces the following output of the program:

CS1110 Computer Science I

Minimum: 32
 Maximum: 100
 Average: 81.53333333333333

Grade Distribution:

00-09:
10-19:
20-29:
30-39: *
40-49:
50-59:
60-69: *
70-79: ***
80-89: *****
90-99: **
100: **

=====

CS1120 Computer Science II

Minimum: 28
Maximum: 100
Average: 71.125

Grade Distribution:

00-09:
10-19:
20-29: **
30-39:
40-49:
50-59: **
60-69: ***
70-79: ***
80-89: ***
90-99: **
100: *

I verified that this output satisfies program requirements.

PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)

No refinements are needed. In this program, I have already included all required features.

PHASE 8: PRODUCTION

I prepared a copy of the entire program for Lab TA’s evaluation, as specified by the TA. Then, I sent electronically the copy to the Lab TA.

PHASE 9: MAINTENANCE

To fully benefit from the program evaluation feedback received from the Lab TA, I will perform program maintenance. This means that I should use all TAs feedback to improve my program.