SWEN 3601
SOFTWARE CONSTRUCTION
FINAL EXAM

Master of Software Engineering
Birzeit University
December 22, 2018 @ 2:00pm → January 3, 2019 @ 5:00pm

This is an OPEN RESOURCES TAKE HOME exam. Textbooks, notes, laptops, calculators, cell phones, and Internet access are allowed. The solution to this exam should be solely your contribution without any other human collaborations. If other resources have been used, please be advised to use proper acknowledgment text.

This exam is worth a total of 30 marks. There are 3 sections, 18 questions, and 7 pages (including this cover page). Please read each question carefully, and submit a PDF document with your solution no later than January 3, 2019 @ 5:00pm. You can submit your solutions on Ritaj or via email at: ahmedtamrawi@gmail.com. No late submissions will be accepted or tolerated. Good luck!

Name: _____

Student I.D.: _____

Section 1: Formula 1 Racing Cars (36 marks)

The `DataProcessor` class is implemented to operate some calculation tasks related to Formula 1 racing cars. Study the following implementation of the class in Listing 1, Listing 2, and Listing 3 to answer the following questions:

Q1. (*6 marks*) Suggest strategies to speed-up the code between the lines $12 - 27$ of function `calculate` in Listing 3.

Q2. (*6 marks*) Perform code jamming for the code between the lines $12 - 33$ of function `calculate` in Listing 3.

Q3. (*6 marks*) Simplify the code between the lines $27 - 34$ in function `loadData` in Listing 2 using a *sentinel value* and discuss possible change(s) for code speed-up.

Q4. (*6 marks*) Introduce look-up table(s) considering the complete class to simplify and speed-up the code. Discuss your intuition for choosing a specific type(s) of look-up table(s) and discuss possible change(s).

Q5. (*6 marks*) Which functions of the class `DataProcessor` do require assertions/exception handling and for which parts of the code? Use line numbers to refer to code in your examples.

Q6. (*6 marks*) Revise the function `loadData` in Listing 2 for better span and live-time for defined variables.

```java
public class DataProcessor extends Formula1 {
    // team, car, sensor
    public double[][][] data;
    public double[][] limit;

    // constructor
    public DataProcessor(double[][][] data, double[][] limit) {
        this.data = data;
        this.limit = limit;
    }

    // calculates average for sensor readings
    private double average(double[] array) {
        int i = 0;
        double val = 0;
        for (i = 0; i < array.length; i++) {
            val += array[i];
        }

        return val / array.length;
    }

    // print sensor average
    private void printSensorAverage(int flag, int sId) {
        if (flag == 0)
            System.out.println(average(data[0][sId]));
        else if (flag == 1)
            System.out.println(average(data[1][sId]));
        else if (flag == 2)
            System.out.println(average(data[2][sId]));
        else if (flag == 3)
            System.out.println(average(data[3][sId]));
        else if (flag == 4)
            System.out.println(average(data[6][sId]));
        else
            System.out.println(average(data[8][sId]));
    }

    // load given data and check value
    private void loadData(String file, double val) {
        // implementation in Listing 2
    }

    // calculate data, always call loadData before calling this function
    public void calculate(double d) {
       // implementation in Listing 3
    }
}
```

Listing 1: Implementation for class `DataProcessor`

```java
private void loadData(String file, double val) {
    int flag = -1;
    boolean found = true;
    int i = 0;
    double[] dataArr;

    BufferedReader in;
    try {
        in = new BufferedReader(new FileReader(file));
        String data = in.readLine();

        StringTokenizer stk = new StringTokenizer(data, "\t");
        dataArr = new double[stk.countTokens()];

        while (stk.hasMoreTokens()) {
            dataArr[i] = Double.parseDouble(stk.nextToken());
            i++;
        }

        in.close();
    } catch (Exception e) {
        System.out.println("Error= " + e);
    }

    i = 0;

    // in a while loop, check whether the value is available
    while ((i < dataArr.length) && (!found)) {
        if (dataArr[i] == val)
            found = true;
        else
            i++;

    }

    if (found) {
        System.out.println("Entry is available");

        if (i < 3)
            flag = 0;
        else if (i < 6)
            flag = 1;
        else if (i < 10)
            flag = 2;
        else if (i < 15)
            flag = 3;
        else if (i < 20)
            flag = 4;
        else if (i < 25)
            flag = 5;

        // print sensor average
        printSensorAverage(flag, 0);
    }
}
```

Listing 2: Code for `loadData` function

```
public void calculate(double d) {

    int i, j, k = 0;
    double[][][] data2 = new double[data.length][data[0].length][data[0][0].length];

    BufferedWriter out;

    // Write racing stats data into a file
    try {
        out = new BufferedWriter(new FileWriter("RacingStatsData.txt"));

        for (i = 0; i < data.length; i++) {
            for (j = 0; j < data[0].length; j++) {
                for (k = 0; k < data[0][0].length; k++) {
                    data2[i][j][k] = data[i][j][k] / d - Math.pow(limit[i][j], 2.0);

                    if (average(data2[i][j]) > 10 && average(data2[i][j]) < 50)
                        break;
                    else if (Math.max(data[i][j][k], data2[i][j][k]) > data[i][j][k])
                        break;
                    else if (Math.pow(Math.abs(data[i][j][k]), 3) < Math.pow(Math.abs(data2
    [i][j][k]), 3) && average(data[i][j]) < data2[i][j][k] && (i + 1) * (j + 1) > 0)
                        data2[i][j][k] *= 2;
                    else
                        continue;
                }
            }
        }

        for (i = 0; i < data2.length; i++) {
            for (j = 0; j < data2[0].length; j++) {
                out.write(data2[i][j] + "\t");
            }
        }

        out.close();

    } catch (Exception e) {
        System.out.println("Error= " + e);
    }
}
```

Listing 3: Code for `calculate` function

Section 2: Apache POI (36 marks)

The Apache POI project[1] provides pure Java libraries for reading and writing files in Microsoft Office formats, such as Word, PowerPoint and Excel. The source code of the project can be found on Github[2]. Specifically, we are interested in the class `XSSFWorkbook`[3]. This class presents a case of poorly written code with severe problems in coupling, cohesion, style, performance, and debugging. It also lacks proper documentation and user comments. Many attempts have been made to document the class through JavaDoc documentation[4] and DynaDoc documentation[5].

Familiarize yourself with the source code for the class `XSSFWorkbook` at the provided link and use the generated documentation (e.g., JavaDoc and DynaDoc documentation) to answer the questions below:

Q1. (*6 marks*) Comment on the class structure complexity, ease of maintainability and extensibility, coupling, cohesion, and resuability. List any issue(s) you found in each aspect and refer to specific line number(s) for each listed issue.

Q2. (*6 marks*) Mention at least two design patterns[6] used in this class along with line numbers. Briefly describe each design pattern.

Q3. (*6 marks*) Discuss at least three different problems on the code style and user comments and suggest proper changes. Refer to the line numbers for each answer. *All examples exposing the same problem will be counted as one.*

Q4. (*6 marks*) Write a high-quality source code version of function `getUniqueSheetName`[7] (lines $692 - 721$).

Q5. (*6 marks*) Comment on the exception and error handling of function `setSheetName`[8] (lines $1588 - 1617$). Are all possible error cases handled? and is the corrective operation on line 1598[9] acceptable and why?

Q6. (*6 marks*) Determine how many test cases required for function `containsSheet`[10] (lines $1806 - 1825$) using *basis path testing* technique [11] and list at least 3 possible test cases.

[1] https://poi.apache.org/
[2] https://github.com/apache/poi
[3] https://github.com/apache/poi/blob/REL_3_17_FINAL/src/ooxml/java/org/apache/poi/xssf/usermodel/XSSFWorkbook.java
[4] https://poi.apache.org/apidocs/dev/org/apache/poi/xssf/usermodel/XSSFWorkbook.html
[5] https://ensoftcorp.github.io/DynaDoc/
[6] https://refactoring.guru/design-patterns/catalog
[7] https://github.com/apache/poi/blob/REL_3_17_FINAL/src/ooxml/java/org/apache/poi/xssf/usermodel/XSSFWorkbook.java#L692
[8] https://github.com/apache/poi/blob/REL_3_17_FINAL/src/ooxml/java/org/apache/poi/xssf/usermodel/XSSFWorkbook.java#L1588
[9] https://github.com/apache/poi/blob/REL_3_17_FINAL/src/ooxml/java/org/apache/poi/xssf/usermodel/XSSFWorkbook.java#L1598
[10] https://github.com/apache/poi/blob/REL_3_17_FINAL/src/ooxml/java/org/apache/poi/xssf/usermodel/XSSFWorkbook.java#L1806
[11] https://www.tutorialspoint.com/software_testing_dictionary/basis_path_testing.htm

Section 3: Software Construction Concepts and Principals (30 marks)

Q1. (*5 marks*) Describe the Model-View-Controller architecture and how it helps in reducing maintenance effort.

Q2. (*5 marks*) Does full structural coverage imply full branch coverage and full path coverage? Explain.

Q3. (*5 marks*) Provide a brief description of any one of the following design patterns: *Abstract Factory*, *Bridge*, or *Iterator*.

Q4. (*5 marks*) Search the web and articulate your findings about the term *Code Technical Debt*, its relation to code complexity and ease of maintenance, and its causes and how to avoid it.

Q5. (*5 marks*) Articulate the understanding you have gained in this course about what is a clean code and the challenge(s) for creating clean code.

Q6. (*5 marks*) Debug the program `sample.c` in Listing 4. Report two sets of inputs: one for which the program produces the correct result and another for which it does not. Give a precise explanation of why the program works incorrectly sometimes but not always. Provide a 1-line fix.

```c
#include <stdio.h>
#include <stdlib.h>
static void shell_sort(int a[], int size) {
    int i, j;
    int h = 1;

    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++) {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}

int main(int argc, char *argv[]) {
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    printf("Output: ");
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");
    free(a);
    return 0;
}
```

Listing 4: `sample.c` file from `https://www.st.cs.uni-saarland.de/whyprogramsfail/code/sample/sample.c`