

ICSI201 Introduction to Computer Science

Project 04 by Qi Wang

The following parts are included in this document:

- Part I: General project information
- Part II: Project grading rubric
- Part III: Project description
- Appendix A: An example (How to complete a project from start to finish?)

Proper use of the course materials:

All course materials including source codes/diagrams, lecture notes, etc., are for your reference only. Any misuse of the materials is prohibited. For example,

- Copy the source codes/diagrams and modify them into the projects.
 - Students are required to submit the **original work**. For each project, every single statement for each source file and every single algorithm or class diagram for each design must be created by the students from scratch.
- Post the source codes, the diagrams, and other materials online.
- Others

Part I: General Project Information

- All work is individual work unless it is notified otherwise.
- Work will be rejected with no credit if
 - The work is not submitted via Duifene.
 - The work is late.
 - The work is partially or entirely written in Chinese.
 - The work is not submitted properly.
 - Not readable, wrong files, not in required format, crashed files, etc.
 - The work is a copy or partial copy of others' work (such as work from another person or the Internet).
- Students must turn in their original work. Any cheating violation will be reported to the college. Students can help others by sharing ideas, but not by allowing others to copy their work.
- Documents to be included/submitted as a zipped folder:
 - An algorithm – a plain text file
 - Java source file(s) with Javadoc style inline comments
 - Supporting files if any (For example, files containing all testing data.)

Lack of any of the required items or programs with errors will result in a low credit or no credit.

- **How to prepare a zipped folder for work submission?**
 - Copy the above-mentioned files into a folder, rename the folder using the following convention:
[Your_first_name][your_last_name]ProjectNumber
 - For example, *JohnSmithProject04*
 - Zip the folder. A zipped file will be created.
 - For example, a file with name *JohnSmithProject04.zip* will be created.
 - Submit the zip file on Duifene.
 - You must submit a project in this format. **Submissions not in the required format may be rejected or will result in a low credit or no credit.**
- **Grades and feedback:** Co-instructors will grade. Feedback and grades for properly submitted work will be posted on Duifene. For questions regarding the feedback or the grade, students should reach out to their co-instructors prior to discussing with the instructor. Students have limited time/days from when a grade is posted to dispute the grade. Check email daily for the grade review notifications sent from the co-instructors. **Any grade dispute request after the dispute period will not be considered.**

Part II: Project grading rubric

Note: Programs with errors will result in a low credit or no credit.

The following contains only a list of basic PIs used for evaluation. General software development criteria and overall performance of the project are also considered into the final project evaluation.

Project 4: (100 points) Performance Indicator (PI)	LEVELS OF PERFORMANCE INDICATORS				Points Earned
	UNSATISFACTORY	DEVELOPING	SATISFACTORY	EXEMPLARY	
PI1: (10 points) Algorithm: <ul style="list-style-type: none">Three algorithms: main, calculate average and assign letter gradeInput, process, and outputReusable solutionEfficient solutionLogical steps written in English only (No implementation details (source codes) should be included.)	None/Not correct at all. (0)	Some logic steps are present for input, process, and output. But there are issues. Implementation details are included. (≤5)	Logic steps are present for input, process, and output. with minor issues. No implementation details are included. (≤7)	All logic steps are present for input, process, and output. without issues. No implementation details are included. (≤10)	
Comments:					
PI2a: (10 points): Implementation: Comments (Javadoc style): <ul style="list-style-type: none">Comments explain the purpose of each part of the program, not how it is coded.All tags are included correctly.Class commentsMethod commentsBlock comments in <i>main</i> (non- Javadoc style)	None/Not correct at all. (0)	Some comments are written properly. (≤5)	Most comments are written properly. (≤7)	All comments are written properly. (≤10)	
Comments:					
PI2b: (10 points): Implementation:	None/Not correct at all. (0)	Some requirements are met with issues. (≤5)	Most requirements are met with minor issues. (≤7)	All requirements are met with no issues. (≤10)	

<ul style="list-style-type: none"> • Full-word (descriptive) variables • No excessive variables/statements • All variables are declared at the beginning of <i>main</i> • Properly formatted code (indentations) • No errors 					
	Comments:				
PI2c: (15 points): Implementation and test: <ul style="list-style-type: none"> • Correct Input (prompt and format) • Correct for all scores • No hard coding • No errors 	None/Not correct at all. (0)	Some requirements are met with issues. (<=7)	Most requirements are met with minor issues. (<=11)	All requirements are met with no issues. (<=15)	
	Comments:				
PI2d: (15 points): Implementation and test: <ul style="list-style-type: none"> • Correct output (description and format) • Correct output on screen • Correct output into file • No hard coding • No errors 	None/Not correct at all. (0)	Some requirements are met with issues. (<=7)	Most requirements are met with minor issues. (<=11)	All requirements are met with no issues. (<=15)	
	Comments:				
PI2e: (40 points): Implementation and test: <ul style="list-style-type: none"> • Correct process • Correct process for average • Correct process for letter grade 	None/Not correct at all. (0)	Some requirements are met with issues. (<=15)	Most requirements are met with minor issues. (<=30)	All requirements are met with no issues. (<=40)	

<ul style="list-style-type: none">• Correct invocations of average and letter grade function in main• Correct use of loops to avoid redundant codes• Constants are used for menu items for readability• No hard coding• No errors					
Comments:					
Overall comments:					
Total Points Earned					
Not submitted as a zipped folder					-10
Total out of 100					

Part III: Project description

A grade report

Write a program to read test scores, process scores, and display a final report or save the report into a text file. A report is generated for a group of 4 students. In the future, this can be extended easily to a group of n students ($n > 0$).

A test score is an **integer** from 0 to 100. There are no extra credits. A letter grade can be assigned as follows:

If an average is 90 or greater, the letter grade A is assigned.
Otherwise, if an average is 80 or greater, the letter grade B is assigned.
Otherwise, if an average is 70 or greater, the letter grade C is assigned.
Otherwise, the letter grade F is assigned.

Users can choose to display a report like this, save the report into a file, or quit.

Test 1	Test 2	Average	Grade
-----	-----	-----	-----
090	059	074.5	C
082	080	081.0	B
039	100	069.5	F
089	092	090.5	A

Sample run:

Welcome to Grade Center!
Enter 1 to generate and display a grade report.
Enter 2 to generate a grade report and save it into a file.
Enter 3 to quit.
Enter a choice: 1

Enter grades for test1 and test2
For test 1,
Enter score 1: 90
Enter score 2: 82
Enter score 3: 39
Enter score 4: 89

For test 2,
Enter score 1: 59
Enter score 2: 80
Enter score 3: 100
Enter score 4: 92

Test 1	Test 2	Average	Grade
-----	-----	-----	-----
090	059	074.5	C
082	080	081.0	B
039	100	069.5	F
089	092	090.5	A

Welcome to Grade Center!
Enter 1 to generate and display a grade report.
Enter 2 to generate a grade report and save it into a file.
Enter 3 to quit.

When the program runs, this menu is displayed. Choice 1 or 2 can be selected for score processing. Choice 3 can be selected to quit the program.

If users choose choice 1, the program prompts for two test scores. A sequence of scores of each test is entered in order. A report is displayed.

The same menu is displayed until choose 3 is selected (quit).

Enter a choice: 2

Enter grades for test1 and test2

For test 1,

Enter score 1: 76

Enter score 2: 89

Enter score 3: 40

Enter score 4: 100

For test 2,

Enter score 1: 56

Enter score 2: 98

Enter score 3: 99

Enter score 4: 80

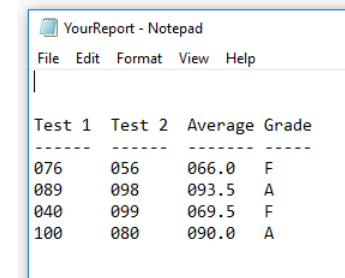
A new file will be created for the report.

Enter the new file name (For example, MyReport.txt.):

YourReport.txt

Report written into a file: YourReport.txt

In this case, choice 2 is selected. After scores are entered, a new file name is entered. A report is written into this file. The file is stored in the root folder of the project. Here is the file.



Test 1	Test 2	Average	Grade
076	056	066.0	F
089	098	093.5	A
040	099	069.5	F
100	080	090.0	A

Welcome to Grade Center!

Enter 1 to generate and display a grade report.

Enter 2 to generate a grade report and save it into a file.

Enter 3 to quit.

Enter a choice: 3

Same menu displays, 3 is selected to quit the program.

Hint:

- You must define two methods for the two common tasks shown below. Use them in *main*.
 - Calculating an average of two integers is a common task. Therefore, a value-returning method can be defined. This method should be a static method, accept two integer arguments and return the average from the method. This method can be called to process the average of the two test scores for each student.
 - Assigning a letter grade based upon the following grading scale is a common task. Therefore, a value-returning method can be defined. This method should be a static method, accept one argument, an average grade, and return a letter grade from the method. This method can be used to assign a letter grade to each student.
 - If an average is 90 or greater, the letter grade A is assigned.
 - Otherwise, if an average is 80 or greater, the letter grade B is assigned.
 - Otherwise, if an average is 70 or greater, the letter grade C is assigned.
 - Otherwise, the letter grade F is assigned.

Notice that three algorithms must be written. Each method must be documented in Javadoc style comments.

- The algorithm of *main*
 - The algorithm of the method calculating an average
 - The algorithm of the method assigning a letter grade
- Instead of writing the same codes for each student, you must reuse the same codes by using loops. For example,
 - how to enter a test one score of a student should be written in a loop, and the loop should execute 4 times for a class of 4 students.
 - how to enter a test two score of a student should be written in a loop, and the loop should execute 4 times for a class of 4 students.
 - how to calculate an average grade should be written in a loop, and the loop should execute 4 times for a class of 4 students.

- how to assign a letter grade for a student should be written in a loop, and the loop should execute 4 times for a class of 4 students.
- how to output a grade report for a student should be written in a loop, and the loop should execute 4 times for a class of 4 students.

- Avoid magic numbers.

In computer programming, a magic number is a special constant used for some specific purpose. It is called magic because its value or presence is inexplicable without some additional knowledge. The term *magic number* also refers to the bad programming practice of using numbers directly in source code without explanation. In most cases this makes programs harder to read, understand, and maintain, although most guides make an exception for the numbers zero and one. For example, a menu is more readable if the menu items are defined as constants as shown in the right column.

Bad practice: using “magic numbers”

Don’t use the following format.

```
switch (choice){
    case 1:
        //
        break;
    case 2:
        //
        break;
    case 3:
        //
        break;
    ...
}
```

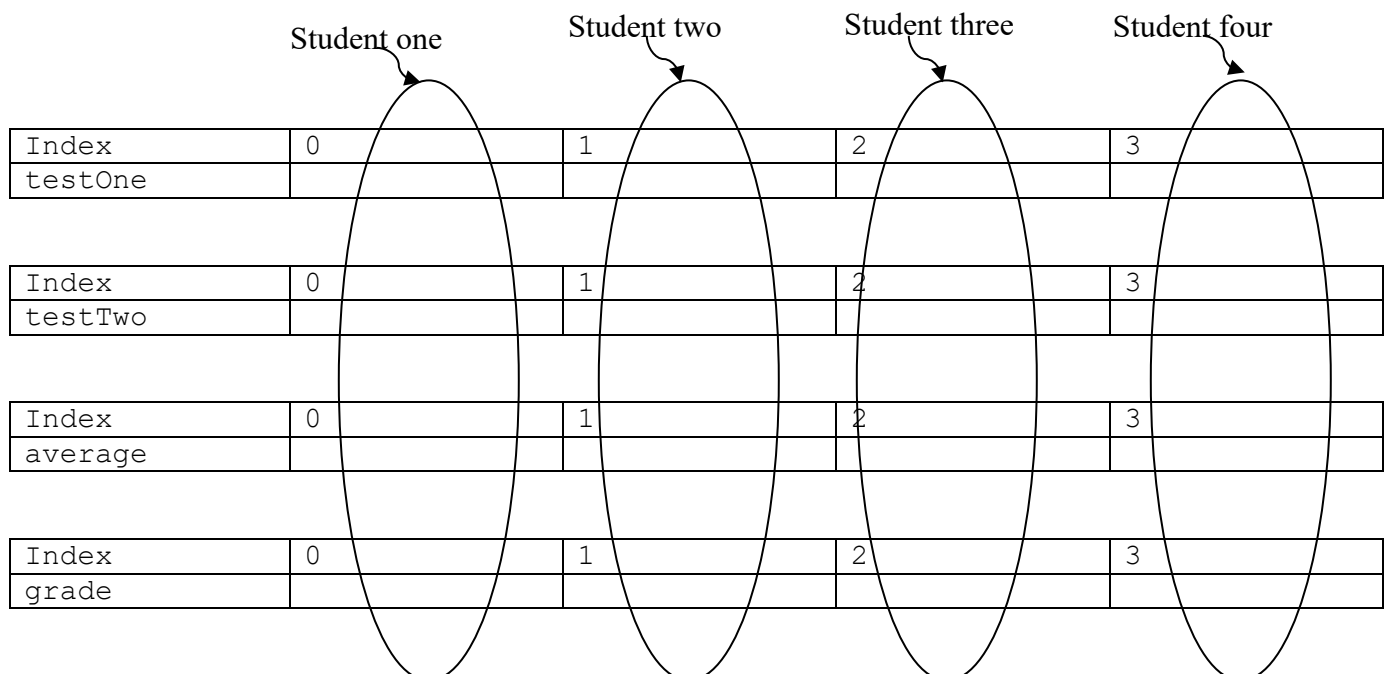
Good practice: more readable using constants

Use the following format.

```
final int GRADE_REPORT = 1;
final int GRADE_REPORT_IN_FILE = 2;
final int EXIT = 3;
```

```
switch (choice){
    case GRADE_REPORT:
        //
        break;
    case GRADE_REPORT_IN_FILE:
        //
        break;
    case EXIT:
        //
        break;
    ...
}
```

- Use four parallel arrays to store corresponding information of each student at the same index. The following shows an example for a class of 4 students.

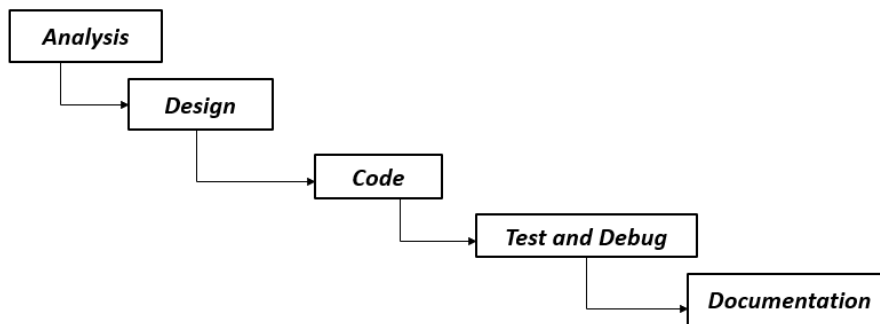


- Access the length of arrays using the instance variable *length*. If an array is full, its length is equal to the number of elements in the array. For example, `testOne.length` returns the size of array `testOne`.

Test your program with many valid input values. Before you get started, I recommend that you study the example included in appendix A.

Appendix A: Example: how to complete a project from start to finish

To complete a project, follow the following program development cycle. These steps are not pure linear but overlapped.



- Read project description to understand all specifications (**Analysis**).
- Create a design (an algorithm for method or a UML class diagram for a class) (**Design**)
- Create a Java program that is a translation of the design. (**Code/Implementation**)
- Test and debug, and (**test/debug**)
- Complete all required documentation. (**Documentation**)

To see the program development activities in action, we will use the following example.

Sample problem:

Write a program to convert a user-entered temperature from Fahrenheit to Celsius. Here is the sample run that shows 0 is displayed after value 32 is entered.

```
Enter a temperature in Fahrenheit: 32
This is equivalent to:
0 Celsius
```

Analysis:

Software must perform to the specifications (meet all customer requirements). The program must satisfy the following requirements:

- It must accept a user-entered temperature in Fahrenheit.
- It must compute and display an equivalent temperature in Celsius.
 - The sample run shows more implementation details.

Design:

For the design, a solution to the problem is created. A solution can be written as an algorithm. An algorithm is a logical series of steps converting inputs to desired outputs. An algorithm consists of three components: *input*, *process*, and *output*.

- Input specifies user-entered input values in order.
- Process specifies how user-entered input values can be converted into desired output values.
- Output specifies how output values can be displayed.

The following solution/algorithm shows how to convert a temperature from Fahrenheit to Celsius.

Summary: Convert a temperature from Fahrenheit to Celsius.

1. Read a temperature in Fahrenheit entered by user.
2. Subtract 32 from the temperature.
3. Multiply the result by 5.
4. And then divide the result by 9.
5. Display the result, the temperature in Celsius.

Input

Process

$$^{\circ}\text{C} = (^{\circ}\text{F} - 32) \times 5 / 9$$

Output

Code:

Implementation is also known as coding. In this activity, an algorithm is translated into a Java program. Add a Java class, add Javadoc [style comments](#) for the class and method *main*. Since *main* is where the algorithm will be translated, the [algorithm](#) can be reused as the starter comments for *main*.

```
/**
 * Temperature conversion from Fahrenheit to Celsius.
 * @author Qi Wang
 * @version 1.0
 */
public class TemperatureConverter{
    /**
     * Reads a temperature in Fahrenheit, and converts it to a temperature in Celsius.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main(String[] args){
        // 1. Read a temperature in Fahrenheit entered by user.
        // 2. Subtract 32 from the temperature.
        // 3. Multiply the result by 5.
        // 4. And then divide the result by 9.
        // 5. Display the result, the temperature in Celsius.
    }
}
```

Proper data types are selected when reserving memory (creating variables) for input values, output values, and important values of process. These values need to be stored in memory so that they can be reused without recalculation. In this case, the input (temperature in Fahrenheit) needs to be stored; the output (temperature in Celsius) needs to be stored. Since the process is very simple, it is not necessary to store the intermediate results from step 2 and 3 of the algorithms. **All variables need to be declared at the beginning of a method for readability and maintainability.**

```
import java.util.Scanner;

/**
 * Temperature conversion from Fahrenheit to Celsius.
 * @author Qi Wang
 * @version 1.0
 */
```

```

public class TemperatureConverter{
    /**
     * Reads a temperature in Fahrenheit and converts it to a temperature in Celsius.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main(String[] args){
        double celsiusTemp, fahrenheitTemp;
        Scanner scan = new Scanner(System.in);
        DecimalFormat temperatureFormat = new DecimalFormat("0.00");

        // 1. Read a temperature in Fahrenheit entered by user.
        // 2. Subtract 32 from the temperature.
        // 3. Multiply the result by 5.
        // 4. And then divide the result by 9.
        // 5. Display the result, the temperature in Celsius.
    }
}

```

Variable	Description
celsiusTemp	Memory location to store the input, temperature in Celsius (double)
fahrenheitTemp	Memory location to store the output, temperature in Fahrenheit (double)
scan	Memory location to store an object reference to a <code>Scanner</code> type object that reads input from keyboard
temperatureFormat	Memory location to store an object reference to a <code>DecimalFormat</code> object that formats a value into a string containing two decimal digits

You may need to declare more variables later. Always add variables to the beginning of the method.

Next, you will translate the algorithm into java code. More requirements including how input should be prompted and entered and how output should be displayed are shown in the sample run. For example, the sample run below

```

Enter a temperature in Fahrenheit: 32
This is equivalent to:
0 Celsius

```

shows additional implementation requirements:

1. The user-entered temperature must be entered at the end of the same line after the exact prompt shown below:
Enter a temperature in Fahrenheit:
2. A line containing **This is equivalent to:** (an exact description) must be inserted as a newline before displaying the output.
3. **Celsius** (an exact description) must be displayed right after the output value.
4. Proper spacing such as one space before the user-entered input (for example, 32) and one space before **Celsius** is required.

A user-entered temperature in Fahrenheit (32), and the equivalent temperature in Celsius (0) can be changed, but the descriptions/prompts (**highlighted text**) are always the same.

See the following added Java codes. Notice that the number of steps in an algorithm is not necessarily equal to the number of programming statements into which they are translated.

```

import java.util.Scanner;

/**
 * Temperature conversion from Fahrenheit to Celsius.
 * @author Qi Wang
 * @version 1.0
 */
public class TemperatureConverter{
    /**
     * Reads a temperature in Fahrenheit, and converts it to a temperature in Celsius.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main(String[] args){
        double celsiusTemp, fahrenheitTemp;
        Scanner scan = new Scanner(System.in);
        DecimalFormat temperatureFormat = new DecimalFormat("0.00");

        //1. Read a temperature in Fahrenheit entered by user.
        System.out.print ("Enter a temperature in Fahrenheit: ");
        fahrenheitTemp = scan.nextDouble();

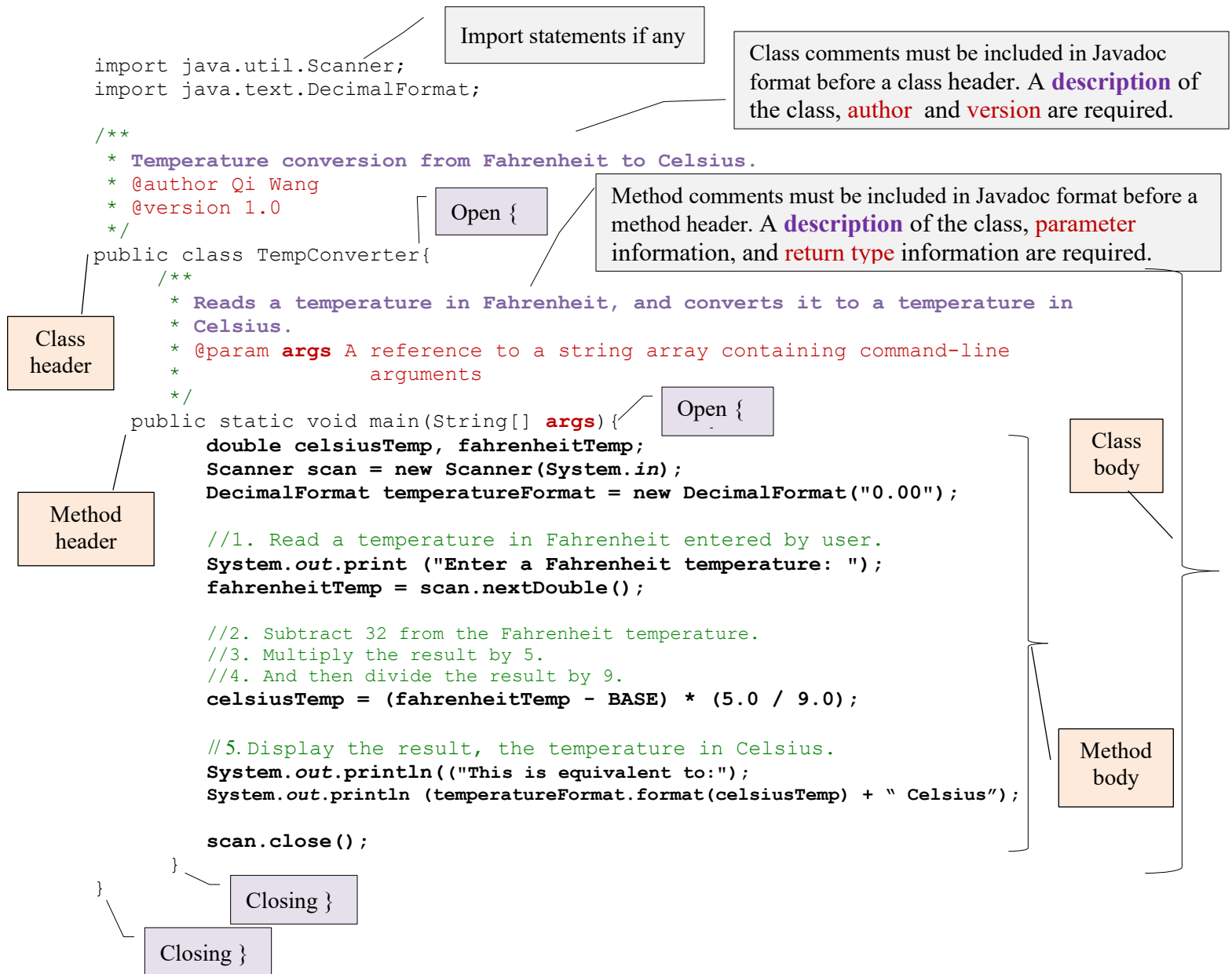
        //2. Subtract 32 from the Fahrenheit temperature.
        //3. Multiply the result by 5.
        //4. And then divide the result by 9.
        celsiusTemp = (fahrenheitTemp - 32) * (5.0 / 9.0);

        //5. Display the result, the temperature in Celsius.
        System.out.println ("This is equivalent to:");
        System.out.println (temperatureFormat.format(celsiusTemp) + " Celsius");

        scan.close();
    }
}

```

More details regarding Javadoc style comments are included on the next page.



To enforce program readability, consistent indentations must be used to show containment hierarchy. Poor readability of a source file can be ignored or misinterpreted. This will affect a software developer's performance. Please read the following. See the example on the next page.

- Align comments with corresponding source codes.
- Use consistent indentations to enhance readability:
 - Indent for the body of a class, the body of each method of a class, and each block of a method.
- Use required spaces, not excess spaces to enhance readability:
 - No spaces between the comments and their corresponding header or code blocks.
 - One space after `*` at the beginning of each line in Javadoc comments. Align first `*` of each line in Javadoc comments.
 - One space in between two adjacent variables.
 - One line in between two segments of codes.

```

import java.util.Scanner;
import java.text.DecimalFormat;

/**
 * Temperature conversion from Fahrenheit to Celsius.
 * @author Qi Wang
 * @version 1.0
 */
public class TempConverter{
    /**
     * Reads a temperature in Fahrenheit entered by user to a temperature in Celsius.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main(String[] args)
    double celsiusTemp, fahrenheitTemp;
    Scanner scan = new Scanner(System.in);
    DecimalFormat temperatureFormat = new DecimalFormat("0.00");

    //1. Read a temperature in Fahrenheit entered by user.
    System.out.print("Enter a Fahrenheit temperature: ");
    fahrenheitTemp = scan.nextDouble();

    //2. Subtract 32 from the Fahrenheit temperature.
    //3. And then, multiply the result by 5.
    //4. And then divide the result by 9.
    celsiusTemp = (fahrenheitTemp - 32) * (5.0 / 9.0);

    //5. Display the result, the temperature in Celsius.
    System.out.println(("This is equivalent to:"));
    System.out.println(temperatureFormat.format(celsiusTemp) + " Celsius");

    scan.close();
}

```

Code conventions are a set of guidelines for a specific programming language that recommend programming style, practices, and methods for each aspect of a piece program written in a programming language. These conventions usually cover file organization, indentation, comments, declarations, statements, white space, naming conventions, programming practices, programming principles, programming rules of thumb, architectural best practices, etc. The following shows some of the conventions for Java.

- A variable name usually starts with a lowercase letter with each of the additional words capitalized (camel case style).
- Remove variables that are never used.
- A class name is usually a singular noun in title case style.
- A constant should be defined as a final variable and should be written in ALL_CAP with an underscore in between two adjacent words. For example, final double SALES_TAX = 0.085;
- Don't use wildcard character * to import all classes from a package. Instead, import classes that are used. One class per import statement.

Debug/Test:

All errors (compile-time errors, run-time errors, and logical errors) must be detected and removed.