ICSI201 Introduction to Computer Science

Project 03 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*, JohnSmithProject03*
  - Zip the folder. A zipped file will be created.
    - For example*,* a file with name *JohnSmithProject03.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 3: (100 points) **Performance Indicator (PI)**	**LEVELS OF PERFORMANCE INDICATORS**				**Points Earned**
	UNSATISFACTORY	**DEVELOPING**	**SATISFACTORY**	**EXEMPLARY**	
PI1: (10 points) **Algorithm:** • Input, process, and output • Reusable solution • Efficient solution • Logical 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):** • Comments explain the purpose of each part of the program, not how it is coded. • All tags are included correctly. • Class comments • Method comments • Block comments in *main* (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:** • Full-word (descriptive) variables • No excessive variables/statements	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)	

• All variables are declared at the beginning of *main* • Properly formatted code (indentations) • No errors					
	Comments:				
PI2c: (15 points): **Implementation and test:** • Correct Input (prompt and format) • Correct for both dates • 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:** • Correct output (description and format) • Correct for year differences • Correct for month differences • 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:** • Correct process • Correct for date differences • No use of date/calendar related classes from Java library • All cases are present • No redundant steps • No hard coding • No errors	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)	
	Comments:				
Overall comments:					

Total Points Earned	
Not submitted as a zipped folder	-10
Total out of 100	

Part III: Project description

A date difference calculator

Write a program to prompt the user for two dates, each of which consists of a month and a year, and display the number of years and months between these two dates. The two dates can be entered in any order. A date difference is described as a nonnegative year difference and a nonnegative month difference.

Sample Run 1:

```
For the first date,
Enter month: August
Enter year: 2011

For the second date,
Enter month: March
Enter year: 1999

These dates are 12 years and 5 months apart.
```

It is not allowed to use any date related Java classes such as *Calendar* or other date related classes. Only arithmetic operations are allowed for calculating date differences.

To calculate a nonnegative year difference and the corresponding nonnegative month difference, subtract the earlier year from the later year and subtract the corresponding months in the same order. This way, a year difference is always nonnegative. But a month difference can be negative. This happens when the years and their months are not in the same order. If so, adjust the month difference by regrouping a year into 12 months. For example, in case 2, the difference is 3 and the initial month difference is -2. This means it would be 3-year difference if this were 2 months later. To adjust, we borrow 1 year difference from the year and add 12 months to the month difference (regrouping a year into 12 months).

Sample run	Dates	Subtraction	After the adjustment
1	August 2011 March 1999	12 years and 5 months	These dates are 12 years and 5 months apart.
2	**June 1999** **April 2002**	**3 years and -2 months**	**These dates are 2 years and 10 months apart.**
3	July 1998 July 1995	3 years and 0 months	These dates are 3 years and 0 months apart.
4	March 1995 July 1995	0 year and 4 months	These dates are 0 years and 4 months apart.
5	December 1998 December 1998	0 year and 0 month	These dates are 0 years and 0 months apart.

Given two dates, the first year can be greater than, less than or equal to the second year. For each case, the first month can be greater than, less than or equal to the second month. Therefore, your solution should address all nine cases.

- First year is greater than second year.
  - First month is greater than second month.
  - First month is less than second month.
  - First month is equal to second month.
- First year is less than second year.
  - First Month is greater than second month.
  - First Month is less than second month.
  - First Month is equal to second month.

- **First year is equal to second year.**
    - **First Month is greater than second month.**
    - **First Month is less than second month.**
    - **First Month is equal to second month.**

In the last three cases when the years are the same, you may simply calculate the absolute value of the month difference.

It is important to store a user-entered month both as a *String* value and as an *int* value (1 for January; 2 for February …). The *String* value is used for month checking; the *int* value is for calculating date difference. Here are more sample runs:

Sample Run 1:
```
For the first date,
Enter month: August
Enter year: 2011

For the second date,
Enter month: March
Enter year: 1999

These dates are 12 years and 5 months apart.
```

Sample Run 2:
```
For the first date,
Enter month: June
Enter year: 1999

For the second date,
Enter month: April
Enter year: 2002

These dates are 2 years and 10 months apart.
```

Sample Run 3:
```
For the first date,
Enter month: July
Enter year: 1998

For the second date,
Enter month: July
Enter year: 1995

These dates are 3 years and 0 months apart.
```

Sample Run 4:
```
For the first date,
Enter month: March
Enter year: 1995

For the second date,
Enter month: July
Enter year: 1995

These dates are 0 years and 4 months apart.
```

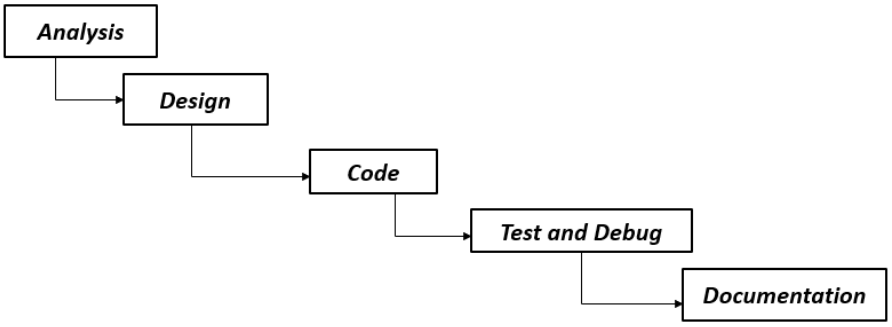
Sample Run 5:
```
For the first date,
Enter month: December
Enter year: 1998
```

```
For the second date,
Enter month: December
Enter year: 1998

These dates are 0 years and 0 months apart.
```

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. **Input**
2. Subtract 32 from the temperature.
3. Multiply the result by 5. **Process** $°C = (°F - 32) \times 5 / 9$
4. And then divide the result by 9.
5. Display the result, the temperature in Celsius. **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 `Scanner` type object that reads input from keyboard
temperatureFormat	Memory location to store an object reference to a `DecimalFormat` 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:

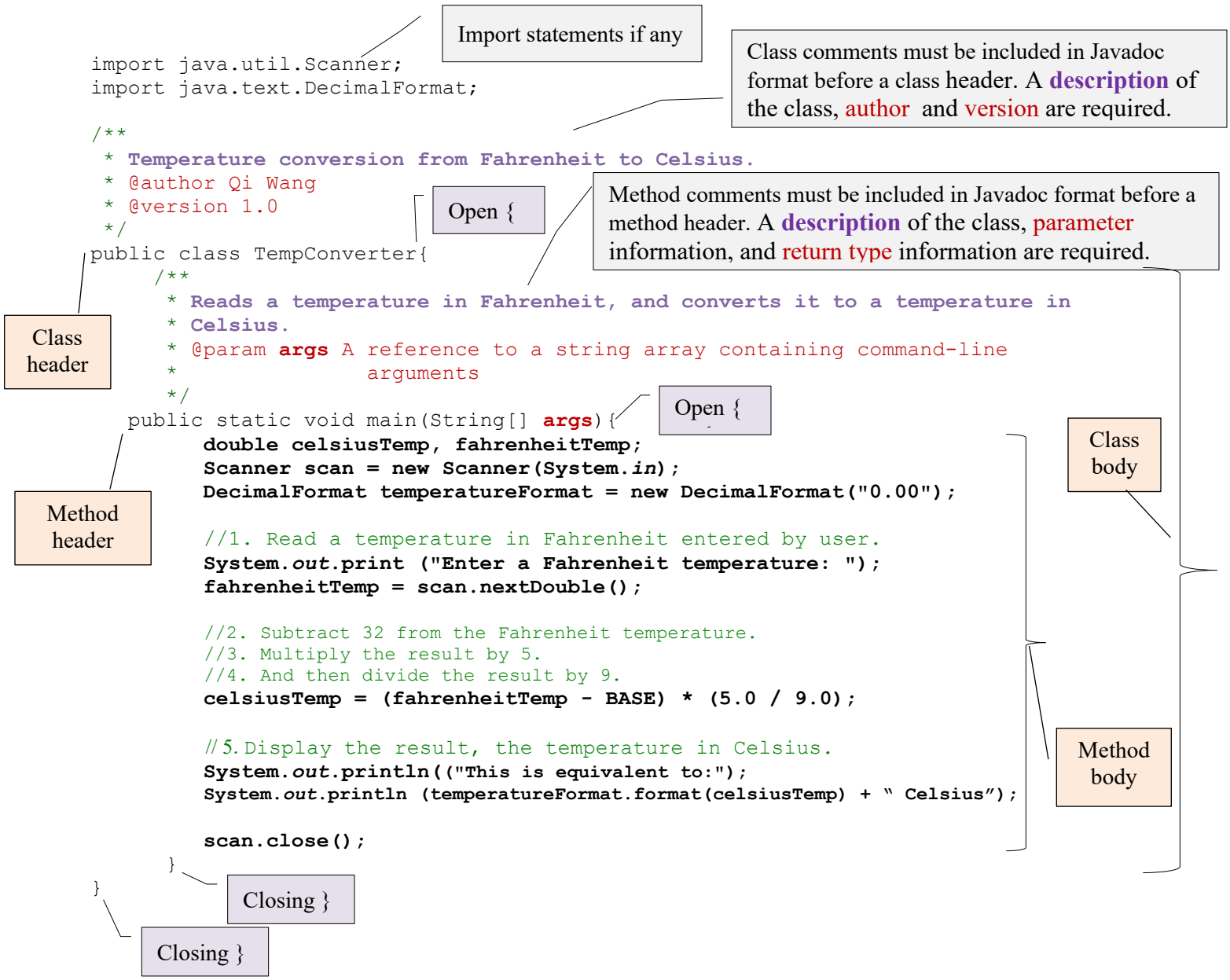
1. The user-entered temperature must be entered at the end of the same line after the exact prompt shown below:
   Enter a temperature in Fahrenheit:
2. A line containing **This is equivalent to: (**an exact description**)** must be inserted as a newline before displaying the output.
3. **Celsius (**an 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.




```
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, 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 Fahrenheit temperature: ");
        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 - BASE) * (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();
    }
}
```

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();
    }
}
```

TAB TAB TAB

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.