

This document has been prepared by Dr. Dmitry Konovalov for James Cook University. Updated 3 August 2017.

© Copyright 2017

This publication is copyright. Apart from any fair dealing for the purpose of private study, research, criticism, or review as permitted under the Copyright Act, no part may be reproduced by any process or placed in computer memory without written permission.

### Instructions for on-campus version:

- **WHEN:** Teaching week #3 at JCU; Teaching week #2 at JCUS/JCUB (scheduled after lectures)
- **DURATION:** two hours
- **ATTENDANCE:** compulsory (students must attend). You (student) **must sign/initial the attendance sheet** provided by your instructor.
- **MARKING [1 mark]:** Complete the tasks from this practical and show the completed tasks to your instructor. Each completed practical is awarded **ONE participation mark** towards the participation assessment component of this subject.
- **EARLY SUBMISSIONS:** You are encouraged to attempt (and complete) some or all of the following tasks **before** attending the practical session.
- **LATE SUBMISSIONS:** You may finish the following tasks in your own time and then show your completed tasks during the following week practical. **The main intent here is to encourage you as much as possible to complete all practicals. If you are late by more than one week**, you will need a valid reason for your instructor to be awarded the marks.

### Instructions for off-campus/online version

- **WHEN:** Teaching week #2 at JCU; Teaching week #1 at JCUS/JCUB (scheduled after lectures)
- **DURATION:** two hours
- **ATTENDANCE:** compulsory (students must attend). You (student) **must submit your work to the appropriate CP2408 Practical assessment dropbox on LearnJCU.**
- **MARKING:** Complete the tasks from this practical and your instructor will provide feedback via assessment rubrics.

## **TASK-1: Chapter-7 Debugging Exercises [10-20 min]**



### *Debugging Exercises*

1. Each of the following files in the Chapter07 folder of your downloadable student files has syntax and/or logic errors. In each case, determine the problem and fix the program. After you correct the errors, save each file using the same filename preceded with *Fix*. For example, DebugSeven1.java will become **FixDebugSeven1.java**.
    - a. DebugSeven1.java
    - b. DebugSeven2.java
    - c. DebugSeven3.java
    - d. DebugSeven4.java
- Fork [https://github.com/CP2406Programming2/cp2406\\_farrell8\\_ch07](https://github.com/CP2406Programming2/cp2406_farrell8_ch07) and then create the corresponding IntelliJ project.
  - Try to build this chapter project. IntelliJ will display **compiling** errors and highlight the offending sections of the java code. Work your way through all of them until all compiling errors are fixed. Run each class [that contains the **main**-function] to see what it does. Commit and push your solution to your github account.

## **TASK-2: Chapter-7 Programming Exercises [10-20 min]**

- Complete any **two** exercises from the following list, **or as directed by your instructor**.
- **Help:** See textbook, and/or [https://github.com/CP2406Programming2/cp2406\\_farrell8\\_ch07](https://github.com/CP2406Programming2/cp2406_farrell8_ch07) (or your own fork where you fixed all compiling errors).
- **MORE HELP:** [https://github.com/CP2406Programming2/cp2306\\_farrell8\\_prac\\_solutions/tree/master/Chapter07/ProgrammingExercises](https://github.com/CP2406Programming2/cp2306_farrell8_prac_solutions/tree/master/Chapter07/ProgrammingExercises).

```

import java.util.Scanner;
public class CharacterInfo
{
    public static void main(String[] args)
    {
        char aChar = 'C';
        System.out.println("The character is " + aChar);
        if(Character.isUpperCase(aChar))
            System.out.println(aChar + " is uppercase");
        else
            System.out.println(aChar + " is not uppercase");
        if(Character.isLowerCase(aChar))
            System.out.println(aChar + " is lowercase");
        else
            System.out.println(aChar + " is not lowercase");
        aChar = Character.toLowerCase(aChar);
        System.out.println("After toLowerCase(), aChar is " + aChar);
        aChar = Character.toUpperCase(aChar);
        System.out.println("After toUpperCase(), aChar is " + aChar);
        if(Character.isLetterOrDigit(aChar))
            System.out.println(aChar + " is a letter or digit");
        else
            System.out.println(aChar +
                " is neither a letter nor a digit");
        if(Character.isWhitespace(aChar))
            System.out.println(aChar + " is whitespace");
        else
            System.out.println(aChar + " is not whitespace");
    }
}

```

**Figure 7-3** The CharacterInfo application

1. Modify the CharacterInfo class shown in Figure 7-3 so that the tested character is retrieved from user input. Save the file as **InputCharacterInfo.java**.
2. Write an application that prompts the user for three first names and concatenates them in every possible two-name combination so that new parents can easily compare them to find the most pleasing baby name. Save the file as **BabyNameComparison.java**.

3. a. Create a program that contains a `String` that holds your favorite movie quote and display the total number of spaces contained in the `String`. Save the file as **CountMovieSpaces.java**.  
 b. Write an application that counts the total number of spaces contained in a movie quote entered by the user. Save the file as **CountMovieSpaces2.java**.
4. Write an application that prompts the user for a password that contains at least two uppercase letters, at least two lowercase letters, and at least two digits. Continuously reprompt the user until a valid password is entered. After each entry, display a message indicating whether the user was successful or the reason the user was not successful. Save the file as **ValidatePassword.java**.
5. Write an application that counts the words in a `String` entered by a user. Words are separated by any combination of spaces, periods, commas, semicolons, question marks, exclamation points, or dashes. Figure 7-17 shows two typical executions. Save the file as **CountWords.java**.

```

C:\Java>java CountWords
Enter a string >> Hello! It's a nice day.
There are 5 words in the string

C:\Java>java CountWords
Enter a string >> I think...therefore I am - I think.
There are 7 words in the string

C:\Java>

```

**Figure 7-17** Two typical executions of the `CountWords` application

6. a. Write an application that accepts three `Strings` from the user and displays one of two messages depending on whether the user entered the `Strings` in alphabetical order without regard to case. Save the file as **Alphabetize.java**.  
 b. Write an application that accepts three `Strings` from the user and displays them in alphabetical order without regard to case. Save the file as **Alphabetize2.java**.
7. Three-letter acronyms are common in the business world. For example, in Java you use the IDE (Integrated Development Environment) in the JDK (Java Development Kit) to write programs used by the JVM (Java Virtual Machine) that you might send over a LAN (local area network). Programmers even use the acronym *TLA* to stand for *three-letter acronym*. Write a program that allows a user to enter three words, and display the appropriate three-letter acronym in all uppercase letters. If the user enters more than three words, ignore the extra words. Save the file as **ThreeLetterAcronym.java**.

### **TASK-3: Chapter-8 Debugging Exercises [10-20 min]**



#### **Debugging Exercises**

1. Each of the following files in the `Chapter08` folder of your downloadable student files has syntax and/or logic errors. In each case, determine the problem and fix the program. After you correct the errors, save each file using the same filename preceded with *Fix*. For example, `DebugEight1.java` will become **FixDebugEight1.java**.
  - a. `DebugEight1.java`
  - b. `DebugEight2.java`
  - c. `DebugEight3.java`
  - d. `DebugEight4.java`
- Fork [https://github.com/CP2406Programming2/cp2406\\_farrell8\\_ch08](https://github.com/CP2406Programming2/cp2406_farrell8_ch08) and then create the corresponding IntelliJ project. Work your way through the project classes until all compiling errors are fixed. Run each class [that contains the **main**-function] to see what it does. Commit and push your solution to your github account.



#### **TASK-4: Chapter-8 Programming Exercises [10-20 min]**

- Complete any **two** exercises from the following list, **or as directed by your instructor**.
- **Help:** See textbook, and/or [https://github.com/CP2406Programming2/cp2406\\_farrell8\\_ch08](https://github.com/CP2406Programming2/cp2406_farrell8_ch08) (or your own fork where you fixed all compiling errors).
- **MORE HELP:**  
[https://github.com/CP2406Programming2/cp2306\\_farrell8\\_prac\\_solutions/tree/master/Chapter08/ProgrammingExercises](https://github.com/CP2406Programming2/cp2306_farrell8_prac_solutions/tree/master/Chapter08/ProgrammingExercises).

1. Write an application that stores 12 integers in an array. Display the integers from first to last, and then display the integers from last to first. Save the file as **TwelveInts.java**.
2. Allow a user to enter any number of double values up to 20. The user should enter 99999 to quit entering numbers. Display an error message if the user quits without entering any numbers; otherwise, display each entered value and its distance from the average. Save the file as **DistanceFromAverage.java**.
3.
  - a. Write an application for Cody's Car Care Shop that shows a user a list of available services: *oil change*, *tire rotation*, *battery check*, or *brake inspection*. Allow the user to enter a string that corresponds to one of the options, and display the option and its price as \$25, \$22, \$15, or \$5, accordingly. Display an error message if the user enters an invalid item. Save the file as **CarCareChoice.java**.
  - b. It might not be reasonable to expect users to type long entries such as "oil change" accurately. Modify the CarCareChoice class so that as long as the user enters the first three characters of a service, the choice is considered valid. Save the file as **CarCareChoice2.java**.
4. Create an application containing an array that stores 10 integers. The application should call five methods that in turn (1) display all the integers, (2) display all the integers in reverse order, (3) display the sum of the integers, (4) display all values less than a limiting argument, and (5) display all values that are higher than the calculated average value. Save the file as **ArrayMethodDemo.java**.
5.
  - a. Write an application that accepts up to 10 Strings, or fewer if the user enters a terminating value. Divide the entered Strings into two lists—one for short Strings that are 10 characters or fewer and the other for long Strings. After data entry is complete, prompt the user to enter which type of String to display, and then output the correct list. For this exercise, you can assume that if the user does not request the list of short strings, the user wants the list of long strings. If there are no Strings in a requested list, output an appropriate message. Prompt the user continuously until a sentinel value is entered. Save the file as **CategorizeStrings.java**.
  - b. Modify the CategorizeStrings application to divide the entered Strings into those that contain no spaces, one space, or more. After data entry is complete, continuously prompt the user to enter the type of String to display. If the user does not enter one of the three valid choices, display all of the Strings. Save the file as **CategorizeStrings2.java**.
6.
  - a. Create a class named Salesperson. Data fields for Salesperson include an integer ID number and a double annual sales amount. Methods include a constructor that requires values for both data fields, as well as get and set methods for each of the

433

data fields. Write an application named `DemoSalesperson` that declares an array of 10 `Salesperson` objects. Set each ID number to 9999 and each sales value to zero. Display the 10 `Salesperson` objects. Save the files as **`Salesperson.java`** and **`DemoSalesperson.java`**.

- b. Modify the `DemoSalesperson` application so each `Salesperson` has a successive ID number from 111 through 120 and a sales value that ranges from \$25,000 to \$70,000, increasing by \$5,000 for each successive `Salesperson`. Save the file as **`DemoSalesperson2.java`**.
7.
  - a. Create a `CollegeCourse` class. The class contains fields for the course ID (for example, "CIS 210"), credit hours (for example, 3), and a letter grade (for example, 'A'). Include get and set methods for each field. Create a `Student` class containing an ID number and an array of five `CollegeCourse` objects. Create a get and set method for the `Student` ID number. Also create a get method that returns one of the `Student`'s `CollegeCourses`; the method takes an integer argument and returns the `CollegeCourse` in that position (0 through 4). Next, create a set method that sets the value of one of the `Student`'s `CollegeCourses`; the method takes two arguments—a `CollegeCourse` and an integer representing the `CollegeCourse`'s position (0 through 4). Save the files as **`CollegeCourse.java`** and **`Student.java`**.
  - b. Write an application that prompts a professor to enter grades for five different courses each for 10 students. Prompt the professor to enter data for one student at a time, including student ID and course data for five courses. Use prompts containing the number of the student whose data is being entered and the course number—for example, "Enter ID for student #s", where *s* is an integer from 1 through 10, indicating the student, and "Enter course ID #n", where *n* is an integer from 1 through 5, indicating the course number. Verify that the professor enters only A, B, C, D, or F for the grade value for each course. Save the file as **`InputGrades.java`**.

=== END OF THIS PRACTICAL ☺ ===