CP2406 Programming-II: Practical-8

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

Instructions for on-campus version:

- **WHEN**: Teaching week **#9** at JCU; Teaching week **#8** 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-12 Exception Handling: Debugging Exercises [10-20 min]

Debugging Exercises

1. Each of the following files in the Chapter12 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, DebugTwelve1.java will become **FixDebugTwelve1.java**. You will also use a file named DebugEmployeeIDException.java with the DebugTwelve4.java file.

   a. DebugTwelve1.java
   b. DebugTwelve2.java
   c. DebugTwelve3.java
   d. DebugTwelve4.java

- Fork https://github.com/CP2406Programming2/cp2406_farrell8_ch12 and then create the corresponding IntelliJ project. 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-13 File Input and Output: Debugging Exercises [10-20 min]

Debugging Exercises

1. Each of the following files in the Chapter13 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, DebugThirteen1.java will become **FixDebugThirteen1.java**.

   a. DebugThirteen1.java
   b. DebugThirteen2.java
   c. DebugThirteen3.java
   d. DebugThirteen4.java

   The Chapter13 folder contains four additional data files named DebugData1.txt, DebugData2.txt, DebugData3.txt, and DebugData4.txt. These files are used by the Debug programs.

- Fork https://github.com/CP2406Programming2/cp2406_farrell8_ch13 and then create the corresponding IntelliJ project. 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-3: Chapter-14 Introduction to Swing Components: Debugging Exercises [10-20 min]

Debugging Exercises

1. Each of the following files in the Chapter14 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, DebugFourteen1.java will become **FixDebugFourteen1.java**.

   a. DebugFourteen1.java
   b. DebugFourteen2.java
   c. DebugFourteen3.java
   d. DebugFourteen4.java

- Fork https://github.com/CP2406Programming2/cp2406_farrell8_ch14 and then create the corresponding IntelliJ project. 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-4: Chapter-14 Introduction to Swing Components: Programming Exercises [10-20 min]

- Complete any ***four*** exercises from the following list, ***or as directed by your instructor.***
- **Help**: See textbook, and/or https://github.com/CP2406Programming2/cp2406_farrell8_ch14 (or your own fork where you fixed all compiling errors).
- **MORE HELP:** https://github.com/CP2406Programming2/cp2306_farrell8_prac_solutions/tree/master/Chapter14/ProgrammingExercises .

Programming Exercises

1. a. Write an application that displays a `JFrame` containing the opening sentence or two from your favorite book. Save the file as **JBookQuote.java**.
   b. Add a button to the frame in the `JBookQuote` program. When the user clicks the button, display the title of the book that contains the quote. Save the file as **JBookQuote2.java**.
2. a. Write an application that instantiates a `JFrame` that contains a `JButton`. Disable the `JButton` after the user clicks it. Save the file as **JFrameDisableButton.java**.

b. Modify the `JFrameDisableButton` program so that the `JButton` is not disabled until the user has clicked at least eight times. At that point, display a `JLabel` that indicates "That's enough!". Save the file as **JFrameDisableButton2.java**.

3. Create an application with a `JFrame` and at least five labels that contain interesting historical facts. Every time the user clicks a `JButton`, remove one of the labels and add a different one. Save the file as **JHistoricalFacts.java**.

4. Write an application for Lambert's Vacation Rentals. Use separate `ButtonGroups` to allow a client to select one of three locations, the number of bedrooms, and whether meals are included in the rental. Assume that the locations are parkside for $600 per week, poolside for $750 per week, or lakeside for $825 per week. Assume that the rentals have one, two, or three bedrooms and that each bedroom over one adds $75 to the base price. Assume that if meals are added, the price is $200 more per rental. Save the file as **JVacationRental.java**.

5. a. Write an application that allows a user to select one of at least five television shows to watch on demand. When the user selects a show, display a brief synopsis. Save the file as **JTVDownload.java**.

   b. Change the `JTVDownload` application to include an editable combo box. Allow the user to type the name of a television show and display an appropriate error message if the desired show is not available. Save the file as **JTVDownload2.java**.

6. Design an application for a pizzeria. The user makes pizza order choices from list boxes, and the application displays the price. The user can choose a pizza size of small ($7), medium ($9), large ($11), or extra large ($14), and one of any number of toppings. There is no additional charge for cheese, but any other topping adds $1 to the base price. Offer at least five different topping choices. Save the file as **JPizza.java**.

7. Write an application that allows a user to select a country from a list box that contains at least seven options. After the user makes a selection, display the country's capital city. Save the file as **JCapitals.java**.

8. Write an application that allows the user to choose insurance options in `JCheckBoxes`. Use a `ButtonGroup` to allow the user to select only one of two insurance types—HMO (health maintenance organization) or PPO (preferred provider organization). Use regular (single) `JCheckBoxes` for dental insurance and vision insurance options; the user can select one option, both options, or neither option. As the user selects each option, display its name and price in a text field; the HMO costs $200 per month, the PPO costs $600 per month, the dental coverage adds $75 per month, and the vision care adds $20 per month. When a user deselects an item, make the text field blank. Save the file as **JInsurance.java**.

9. a. Search the Java Web site for information on how to use a `JTextArea`, its constructors, and its `setText()` and `append()` methods. Write an application that allows the user to select options for a dormitory room. Use `JCheckBoxes` for options such as private room, Internet connection, cable TV connection, microwave, refrigerator, and so on. When the application starts, use a text area to display a message listing the options that are not yet selected. As the user selects

and deselects options, add appropriate messages to the common text area so it accumulates a running list that reflects the user's choices. Save the file as **JDorm.java**.

b. Modify the `JDorm` application so that instead of a running list of the user's choices, the application displays only the current choices. Save the file as **JDorm2.java**.

10. Create an application for Paula's Portraits, a photography studio. The application allows users to compute the price of a photography session. Paula's base price is $40 for an in-studio photo session with one person. The in-studio fee is $75 for a session with two or more subjects, and $95 for a session with a pet. A $90 fee is added to take photos on location instead of in the studio. Include a set of mutually exclusive check boxes to select the portrait subject and another set of mutually exclusive check boxes for the session location. Include labels as appropriate to explain the application's functionality. Save the file as **JPhotoFrame.java**.

=== END OF THIS PRACTICAL ☺ ===