CP2406 Programming-II: Practical-7

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

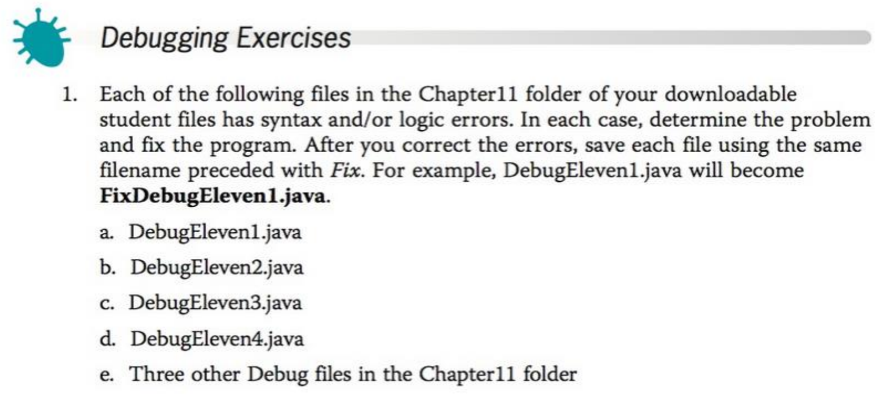
Instructions for on-campus version:

- **WHEN**: Teaching week **#8** at JCU; Teaching week **#7** 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-11 Debugging Exercises [10-20 min]



Debugging Exercises

1. Each of the following files in the Chapter11 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, DebugEleven1.java will become **FixDebugEleven1.java**.

   a. DebugEleven1.java

   b. DebugEleven2.java

   c. DebugEleven3.java

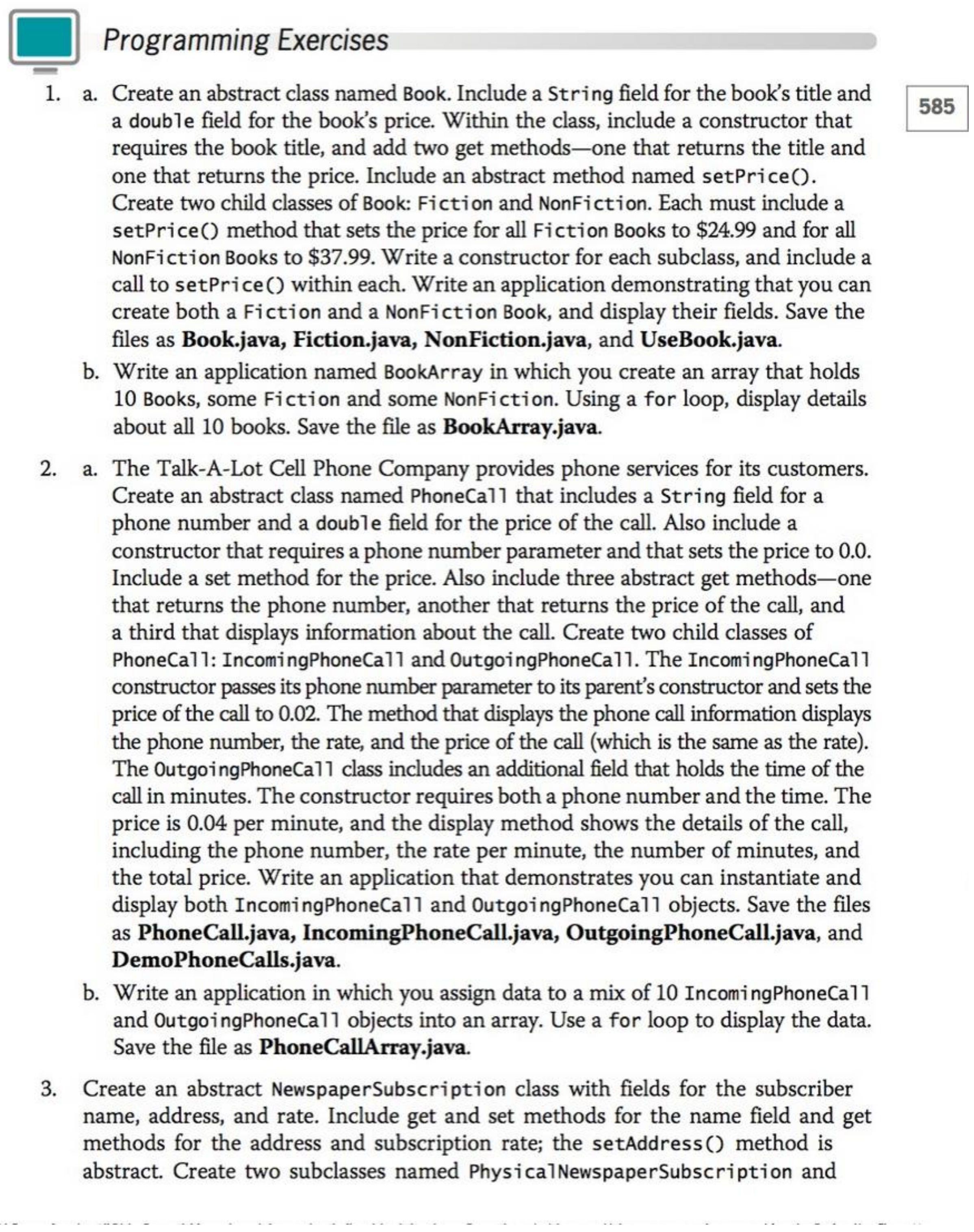
   d. DebugEleven4.java

   e. Three other Debug files in the Chapter11 folder

- Fork https://github.com/CP2406Programming2/cp2406_farrell8_ch11 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-11 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_ch11 (or your own fork where you fixed all compiling errors).
- **MORE HELP:** https://github.com/CP2406Programming2/cp2306_farrell8_prac_solutions/tree/master/Chapter11/ProgrammingExercises .



Programming Exercises

1. a. Create an abstract class named `Book`. Include a `String` field for the book's title and a `double` field for the book's price. Within the class, include a constructor that requires the book title, and add two get methods—one that returns the title and one that returns the price. Include an abstract method named `setPrice()`. Create two child classes of `Book`: `Fiction` and `NonFiction`. Each must include a `setPrice()` method that sets the price for all `Fiction` `Books` to $24.99 and for all `NonFiction` `Books` to $37.99. Write a constructor for each subclass, and include a call to `setPrice()` within each. Write an application demonstrating that you can create both a `Fiction` and a `NonFiction` `Book`, and display their fields. Save the files as **Book.java, Fiction.java, NonFiction.java**, and **UseBook.java**.

   b. Write an application named `BookArray` in which you create an array that holds 10 `Books`, some `Fiction` and some `NonFiction`. Using a `for` loop, display details about all 10 books. Save the file as **BookArray.java**.

2. a. The Talk-A-Lot Cell Phone Company provides phone services for its customers. Create an abstract class named `PhoneCall` that includes a `String` field for a phone number and a `double` field for the price of the call. Also include a constructor that requires a phone number parameter and that sets the price to 0.0. Include a set method for the price. Also include three abstract get methods—one that returns the phone number, another that returns the price of the call, and a third that displays information about the call. Create two child classes of `PhoneCall`: `IncomingPhoneCall` and `OutgoingPhoneCall`. The `IncomingPhoneCall` constructor passes its phone number parameter to its parent's constructor and sets the price of the call to 0.02. The method that displays the phone call information displays the phone number, the rate, and the price of the call (which is the same as the rate). The `OutgoingPhoneCall` class includes an additional field that holds the time of the call in minutes. The constructor requires both a phone number and the time. The price is 0.04 per minute, and the display method shows the details of the call, including the phone number, the rate per minute, the number of minutes, and the total price. Write an application that demonstrates you can instantiate and display both `IncomingPhoneCall` and `OutgoingPhoneCall` objects. Save the files as **PhoneCall.java, IncomingPhoneCall.java, OutgoingPhoneCall.java**, and **DemoPhoneCalls.java**.

   b. Write an application in which you assign data to a mix of 10 `IncomingPhoneCall` and `OutgoingPhoneCall` objects into an array. Use a `for` loop to display the data. Save the file as **PhoneCallArray.java**.

3. Create an abstract `NewspaperSubscription` class with fields for the subscriber name, address, and rate. Include get and set methods for the name field and get methods for the address and subscription rate; the `setAddress()` method is abstract. Create two subclasses named `PhysicalNewspaperSubscription` and

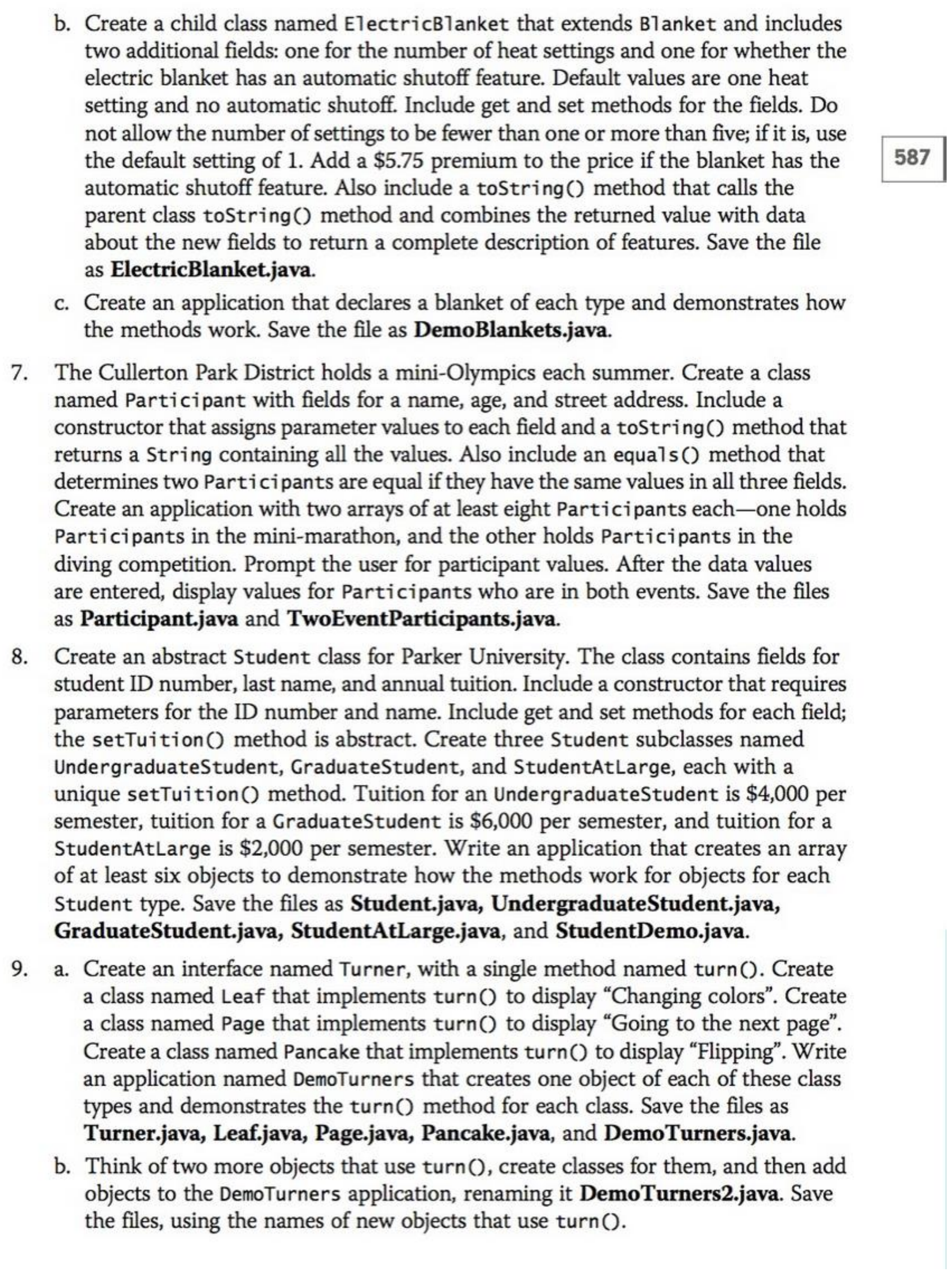
`OnlineNewspaperSubscription`. The parameter for the `setAddress()` method of the `PhysicalNewspaperSubscription` class must contain at least one digit; otherwise, an error message is displayed and the subscription rate is set to 0. If the address is valid, the subscription rate is assigned $15. The parameter for the `setAddress()` method of the `OnlineNewspaperSubscription` class must contain an at sign (@) or an error message is displayed. If the address is valid, the subscription rate is assigned $9. Finally, write an application that declares several objects of both subscription subtypes and displays their data fields. Save the files as **NewspaperSubscription.java, PhysicalNewspaperSubscription.java, OnlineNewspaperSubscription.java**, and **DemoSubscriptions.java**.

4. Create an abstract `Division` class with fields for a company's division name and account number, and an abstract `display()` method. Use a constructor in the superclass that requires values for both fields. Create two subclasses named `InternationalDivision` and `DomesticDivision`. The `InternationalDivision` includes a field for the country in which the division is located and a field for the language spoken; its constructor requires both. The `DomesticDivision` includes a field for the state in which the division is located; a value for this field is required by the constructor. Write an application named `UseDivision` that creates `InternationalDivision` and `DomesticDivision` objects for two different companies and displays information about them. Save the files as **Division.java, InternationalDivision.java, DomesticDivision.java**, and **UseDivision.java**.

5. Create an abstract class named `Element` that holds properties of elements, including their symbol, atomic number, and atomic weight. Include a constructor that requires values for all three properties and a get method for each value. (For example, the symbol for carbon is C, its atomic number is 6, and its atomic weight is 12.01. You can find these values by reading a periodic table in a chemistry reference or by searching the Web.) Also include an abstract method named `describeElement()`.

   Create two extended classes named `MetalElement` and `NonMetalElement`. Each contains a `describeElement()` method that displays the details of the element and a brief explanation of the properties of the element type. For example, metals are good conductors of electricity, while nonmetals are poor conductors. Write an application named `ElementArray` that creates and displays an array that holds at least two elements of each type. Save the files as **Element.java, MetalElement.java, NonMetalElement.java**, and **ElementArray.java**.

6. a. Create a class named `Blanket` with fields for a blanket's size, color, material, and price. Include a constructor that sets default values for the fields as *Twin, white, cotton*, and *$30.00*. Include a set method for each of the first three fields. The method that sets size adds $10 to the base price for a double blanket, $25 for a queen blanket, and $40 for a king. The method that sets the material adds $20 to the price for wool and $45 to the price for cashmere. In other words, the price for a king-sized cashmere blanket is $115. Whenever the size or material is invalid, reset the blanket to the default values. Include a `toString()` method that returns a description of the blanket. Save the file as **Blanket.java**.



b. Create a child class named `ElectricBlanket` that extends `Blanket` and includes two additional fields: one for the number of heat settings and one for whether the electric blanket has an automatic shutoff feature. Default values are one heat setting and no automatic shutoff. Include get and set methods for the fields. Do not allow the number of settings to be fewer than one or more than five; if it is, use the default setting of 1. Add a $5.75 premium to the price if the blanket has the automatic shutoff feature. Also include a `toString()` method that calls the parent class `toString()` method and combines the returned value with data about the new fields to return a complete description of features. Save the file as **ElectricBlanket.java**.

c. Create an application that declares a blanket of each type and demonstrates how the methods work. Save the file as **DemoBlankets.java**.

7. The Cullerton Park District holds a mini-Olympics each summer. Create a class named `Participant` with fields for a name, age, and street address. Include a constructor that assigns parameter values to each field and a `toString()` method that returns a `String` containing all the values. Also include an `equals()` method that determines two `Participants` are equal if they have the same values in all three fields. Create an application with two arrays of at least eight `Participants` each—one holds `Participants` in the mini-marathon, and the other holds `Participants` in the diving competition. Prompt the user for participant values. After the data values are entered, display values for `Participants` who are in both events. Save the files as **Participant.java** and **TwoEventParticipants.java**.

8. Create an abstract `Student` class for Parker University. The class contains fields for student ID number, last name, and annual tuition. Include a constructor that requires parameters for the ID number and name. Include get and set methods for each field; the `setTuition()` method is abstract. Create three `Student` subclasses named `UndergraduateStudent`, `GraduateStudent`, and `StudentAtLarge`, each with a unique `setTuition()` method. Tuition for an `UndergraduateStudent` is $4,000 per semester, tuition for a `GraduateStudent` is $6,000 per semester, and tuition for a `StudentAtLarge` is $2,000 per semester. Write an application that creates an array of at least six objects to demonstrate how the methods work for objects for each `Student` type. Save the files as **Student.java, UndergraduateStudent.java, GraduateStudent.java, StudentAtLarge.java**, and **StudentDemo.java**.

9. a. Create an interface named `Turner`, with a single method named `turn()`. Create a class named `Leaf` that implements `turn()` to display "Changing colors". Create a class named `Page` that implements `turn()` to display "Going to the next page". Create a class named `Pancake` that implements `turn()` to display "Flipping". Write an application named `DemoTurners` that creates one object of each of these class types and demonstrates the `turn()` method for each class. Save the files as **Turner.java, Leaf.java, Page.java, Pancake.java**, and **DemoTurners.java**.

   b. Think of two more objects that use `turn()`, create classes for them, and then add objects to the `DemoTurners` application, renaming it **DemoTurners2.java**. Save the files, using the names of new objects that use `turn()`.

=== END OF THIS PRACTICAL ☺ ===