CP2406 Programming-II: Practical-6

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 **#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-10 Debugging Exercises [10-20 min]

Debugging Exercises

1. Each of the following files in the Chapter10 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, DebugTen1.java will become **FixDebugTen1.java**.
   a. DebugTen1.java
   b. DebugTen2.java
   c. DebugTen3.java
   d. DebugTen4.java
   e. Eight other Debug files are available in the Chapter10 folder; these files are used by the DebugTen exercises.

- Fork https://github.com/CP2406Programming2/cp2406_farrell8_ch10 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-10 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_ch10 (or your own fork where you fixed all compiling errors).
- **MORE HELP:** https://github.com/CP2406Programming2/cp2306_farrell8_prac_solutions/tree/master/Chapter10/ProgrammingExercises .

1. Create a class named `Horse` that contains data fields for the name, color, and birth year. Include get and set methods for these fields. Next, create a subclass named `RaceHorse`, which contains an additional field that holds the number of races in which the horse has competed and additional methods to get and set the new field. Write an application that demonstrates using objects of each class. Save the files as **Horse.java, RaceHorse.java**, and **DemoHorses.java**.
2. Mick's Wicks makes candles in various sizes. Create a class for the business named `Candle` that contains data fields for `color`, `height`, and `price`. Create get methods for all three fields. Create set methods for `color` and `height`, but not for `price`. Instead, when `height` is set, determine the `price` as $2 per inch. Create a child class named `ScentedCandle` that contains an additional data field named `scent` and methods to get and set it. In the child class, override the parent's `setHeight()` method to set the price of a `ScentedCandle` object at $3 per inch. Write an application that instantiates an object of each type and displays the details. Save the files as **Candle.java, ScentedCandle.java**, and **DemoCandles.java**.
3. Create an `ItemSold` class for Francis Pet Supply. Fields include an invoice number, description, and price. Create get and set methods for each field. Create a subclass named `PetSold` that descends from `ItemSold` and includes three Boolean fields that indicate whether the pet has been vaccinated, neutered, and housebroken, and include get and set methods for these fields. Write an application that creates two objects of each class, and demonstrate that all the methods work correctly. Save the files as **ItemSold.java, PetSold.java**, and **DemoItemsAndPets.java**.
4. Create a class named `Poem` that contains fields for the name of the poem and the number of lines in it. Include a constructor that requires values for both fields. Also include get methods to retrieve field values. Create three subclasses: `Couplet`, `Limerick`, and `Haiku`. The constructor for each subclass requires only a title; the lines field is set using a constant value. A couplet has two lines, a limerick has five lines, and a haiku has three lines. Create an application that demonstrates usage of an object of each type. Save the files as **Poem.java, Couplet.java, Limerick.java, Haiku.java**, and **DemoPoems.java**.
5. The developers of a free online game named Sugar Smash have asked you to develop a class named `SugarSmashPlayer` that holds data about a single player. The class contains the following fields: the player's integer ID number, a `String` screen name, and an array of integers that stores the highest score achieved in each of 10 game levels. Include get and set methods for each field. The get and set methods for the scores should each require two parameters—one that represents the score

achieved and one that represents the game level to be retrieved or assigned. Display an error message if the user attempts to assign or retrieve a score from a level that is out of range for the array of scores. Additionally, no level except the first one should be set unless the user has earned at least 100 points at each previous level. If a user tries to set a score for a level that is not yet available, issue an error message. Create a class named `PremiumSugarSmashPlayer` that descends from `SugarSmashPlayer`. This class is instantiated when a user pays $2.99 to have access to 40 additional levels of play. As in the free version of the game, a user cannot set a score for a level unless the user has earned at least 100 points at all previous levels. Create a program that instantiates several objects of each type and demonstrates the methods. Save the files as **SugarSmashPlayer.java, PremiumSugarSmashPlayer.java**, and **DemoSugarSmash.java**.

6. Create a class named `BaseballGame` that contains data fields for two team names and scores for each team in each of nine innings. Create get and set methods for each field; the get and set methods for the scores should require a parameter that indicates which inning's score is being assigned or retrieved. Do not allow an inning score to be set if all the previous innings have not already been set. If a user attempts to set an inning that is not yet available, issue an error message. Also include a method that determines the winner of the game after scores for the last inning have been entered. (For this exercise, assume that a game might end in a tie.) Create two subclasses from `BaseballGame`: `HighSchoolBaseballGame` and `LittleLeagueBaseballGame`. High school baseball games have seven innings, and Little League games have six innings. Ensure that scores for later innings cannot be accessed for objects of these subtypes. Write three applications that each instantiate one of the object types and demonstrate their methods. Save the files as **BaseballGame.java, HighSchoolBaseballGame.java, LittleLeagueBaseballGame.java, DemoBaseballGame.java, DemoHSBaseballGame.java**, and **DemoLLBaseballGame.java**.

7. Create a class named `Package` with data fields for weight in ounces, shipping method, and shipping cost. The shipping method is a character: *A* for air, *T* for truck, or *M* for mail. The `Package` class contains a constructor that requires arguments for weight and shipping method. The constructor calls a `calculateCost()` method that determines the shipping cost, based on the following table:

Weight (oz.)	Air ($)	Truck ($)	Mail ($)
1 to 8	2.00	1.50	.50
9 to 16	3.00	2.35	1.50
17 and over	4.50	3.25	2.15

The `Package` class also contains a `display()` method that displays the values in all four fields. Create a subclass named `InsuredPackage` that adds an insurance cost to the shipping cost, based on the following table:

Shipping Cost Before Insurance ($)	Additional Cost ($)
0 to 1.00	2.45
1.01 to 3.00	3.95
3.01 and over	5.55

Write an application named `UsePackage` that instantiates at least three objects of each type (`Package` and `InsuredPackage`) using a variety of weights and shipping method codes. Display the results for each `Package` and `InsuredPackage`. Save the files as **Package.java, InsuredPackage.java**, and **UsePackage.java**.

8. Create a class named `CollegeCourse` that includes data fields that hold the department (for example, ENG), the course number (for example, 101), the credits (for example, 3), and the fee for the course (for example, $360). All of the fields are required as arguments to the constructor, except for the fee, which is calculated at $120 per credit hour. Include a `display()` method that displays the course data. Create a subclass named `LabCourse` that adds $50 to the course fee. Override the parent class `display()` method to indicate that the course is a lab course and to display all the data. Write an application named `UseCourse` that prompts the user for course information. If the user enters a class in any of the following departments, create a `LabCourse`: BIO, CHM, CIS, or PHY. If the user enters any other department, create a `CollegeCourse` that does not include the lab fee. Then display the course data. Save the files as **CollegeCourse.java, LabCourse.java**, and **UseCourse.java**.

9. Create a class named `Rock` that acts as a superclass for rock samples collected and catalogued by a natural history museum. The `Rock` class contains fields for a number of samples, a description of the type of rock, and the weight of the rock in grams. Include a constructor that accepts parameters for the sample number and weight. The `Rock` constructor sets the description value to *Unclassified*. Include get methods for each field. Create three child classes named `IgneousRock`, `SedimentaryRock`, and `MetamorphicRock`. The constructors for these classes require parameters for the sample number and weight. Search the Web for a brief description of each rock type and assign it to the description field. Create an application that instantiates an object of each type and demonstrate that the methods work appropriately. Save the files as **Rock.java, IgneousRock.java, SedimentaryRock.java, MetamorphicRock.java**, and **DemoRocks.java**.

10. Develop a set of classes for a college to use in various student service and personnel applications. Classes you need to design include the following:

    - `Person`—A `Person` contains a first name, last name, street address, zip code, and phone number. The class also includes a method that sets each data field, using a series of dialog boxes and a display method that displays all of a `Person`'s information on a single line at the command line on the screen.

=== END OF THIS PRACTICAL ☺ ===