Program #1

Due: **9/21/18 at 10:00 PM**
Late Collections: **9/22/18 at 10:00PM**
9/23/18 at 10:00 PM
Program Files (3): **Couple.java, Person.java, TaxTable.java**

Imagine that you work for a personal accounting firm and have been asked to write a program that is able to do calculations using both the 2017 and 2018 United States tax codes. To simply the problem, we will only focus on taxes for married couples. This assignment allows you to practice your understanding of object-oriented design, especially the principles of data encapsulation and aggregation. You will write three classes: **Person** will represent an individual and their income. **Couple** will represent two people who are married. **TaxTable** will represent a set of tax brackets for a particular condition in a particular year. The requirements for each class are listed below. Unless stated otherwise, use good object-oriented design to determine which methods and fields should be public vs. private and whether to use instance vs. static. To avoid losing style points, use **this** when appropriate.

The **Person** class should have the following methods and fields:

- A string for the person's **name**
- An int for their annual **income,** to be measured in dollars.
- A constructor with two formal parameters: **name** and **income.**
- Get methods for both fields.
- A set method for **income.**

The **Couple** class should have the following methods and fields:

- A **firstSpouse** field and a **secondSpouse** field. Both are of type **Person.**
- A constructor that takes two formal parameters, each of type **Person.**
- Get methods for the **firstSpouse** and **secondSpouse**
- A **getTotalIncome()** method that returns the sum of the incomes of the spouses.
- A **toString()** method that returns a string consisting of the first spouse's name, the character '&', the second spouse's name, the character ':', the first spouse's income in dollars, the character '/' and the second spouse's income. E.g., "Amy & Brian: $50000 / $55000".
- A **calculateSavings()** method that takes two parameters: a base tax table and a comparison tax table. The method returns a double that is the amount of money the couple will save if they pay taxes under the comparison table as opposed to the base table. If instead the base table results in less tax, it returns a negative number.

The **TaxTable** class should have the following methods and fields:

- An instance field called **description** that provides a textual description of the table.
- An array of integers called **incomeLevels** that represents the boundaries of the income levels in the tax table. The first element will always be 0 (the lower bound on incomes). Each of the other values is the top income for a specific tax bracket, given in ascending order.
- An array of doubles called **rates** that represents the tax rate for incomes at each bracket, given in the same order as the bounds of the incomeLevels. Thus, a rate at index *i* refers to the taxes for incomes greater than the incomeLevel at index *i* up to and including those at *i*+1. The last rate is for any income that exceeds the top bound in incomeLevels.
- A boolean field called **jointFile** that is true when the tax table is for married couples filing jointly.
- A constructor that takes three formal parameters: a string description of the table, an array of integers describing the income levels, and an array of doubles describing the tax rates for those income levels. The **jointFile** field is set to **false.**

- A constructor that takes four formal parameters: the three parameters listed in the previous constructor plus a boolean indicating if this a table for joint filers.
- A get method for **description**
- A **calculateTax()** instance method that takes a single parameter giving the annual **income** being taxed. It returns a double that is the corresponding taxes owed in dollars. The United States uses a gradual tax schedule, which means that not all of your income is taxed at the highest rate. Instead the portion of your income that falls in the first bracket is taxed at that rate, the additional portion that falls in the next bracket would be taxed at that rate, and so on. For example, if the brackets were \$0-\$10,000 at 10%, \$10,001-\$30,000 at 15% and \$30,000 - \$60,000 at 20%, then a person earning \$38,000 would pay \$10,000(0.1) + \$20,000(0.15) + \$8,000(0.2) = \$1000 + \$3000 + \$1600 = \$5600.
- A **calculateTax()** instance method that takes a Couple object as a parameter and returns a double value that is the taxes the couple owes according to the current TaxTable object. If the TaxTable is a **jointFile** table, then the sum of their incomes is used to determine the bracket(s) and tax rates. If it not a **jointFile** table, then each individual's income is used to calculate their personal tax, and the two tax amounts are summed to return the total tax for the couple.
- A method **printTaxTables()** that takes two TaxTable objects and prints them to the screen side-by-side. See the sample output on p. 4 for the format of this output. In particular, the description of each table should appear at the top, there should be a line of hyphens and then one line for each bracket with the range and tax rate in neat columns. You will have to think about how to order your code because once you've printed a line, you cannot go back and more at the end of the line. Also, consider building a range string before outputting it, so you can take advantage of printf().
- A method **sortBySavings()** that takes an array of type **Couple**, a base **TaxTable** and a comparison **TaxTable.** Using selectin sort, the method sorts the couples array by the amount of money they would save under the comparison tax table (as opposed to the base tax table), in ascending order. Be thoughtful as you write this method. The value you are sorting on is the result of the **calculateSavings()** method call, but you are actually changing the locations of Couple objects in the array. This will require you to adapt the standard selection sort algorithm.

Write a **main** method in **TaxTable** such that when your program is run, it does the following:

- Creates an array of four tax table objects, where there is one object for each tax table given below:

2017 Married Filing Separately	
\$0 - \$9,325	10%
\$9,326 - \$37,950	15%
\$37,951 - \$76,550	25%
\$76,551 - \$116,675	28%
\$116,676 - \$208,350	33%
\$208,351 - \$235,250	35%
\$235,351+	39.6%

2017 Married Filing Jointly	
\$0 - \$18,650	10%
\$18,651 - \$75,900	15%
\$75,901 - \$153,100	25%
\$153,101 - \$233,350	28%
\$233,351 - \$416,700	33%
\$416,701 - \$470,700	35%
\$470,701+	39.6%

2018 Married Filing Separately	
\$0 - \$9,525	10%
\$9,526 - \$38,700	12%
\$38,701 - \$82,500	22%
\$82,501 - \$157,500	24%
\$157,501 - \$200,000	32%
\$200,001 - \$300,000	35%
\$300,000+	37%

2018 Married Filing Jointly	
\$0 - \$19,050	10%
\$19,051- \$77,400	12%
\$77,401 - \$165,000	22%
\$165,001 - \$315,000	24%
\$315,001 - \$400,000	32%
\$400,001 - \$600,000	35%
\$600,001+	37%

- Print the 2017 tables side-by-side, and then print the 2018 tables side-by-side.

- Create an array of **Couple** objects where the objects are described by the following table:

First spouse		**Second Spouse**	
Name	**Income**	**Name**	**Income**
Michelle	$50,000	Joe	$25,000
Bob	$20,000	Theresa	$0
Gary	$21,000,000	Lisa	$50,000
Henry	$140,000	Ray	$90,000

- For each table, print its description and the amount of taxes Michelle and Joe would owe on consecutive lines. Make sure the results form neatly ordered columns.
- Sort the couples by ~~descending~~ order of the amount they will save if filing jointly in 2018, as opposed to filing jointly under the 2017 tax table.
- Print out the (now sorted) list of couples and their savings from the previous step, **but in descending order,** using each couple's **toString()** method. Make sure that the results form neatly ordered columns.

Hints:

- Recall that Scanner's next() method return a String. Assuming the string **s** is a single character, you can retrieve this character using s.charAt(0).
- When printing numeric quantities, you should consider using the System.out.printf() method in order to get your results to print nicely. When specifying format strings, recall that **d** is used for integers and **f** is used for numbers with a decimal point. With the latter, you can specify a total width and number of decimal places.
- You can also use printf() to ensure that strings have a fixed column width. Just use the **s** format specifier; if you want your string to be left-justified instead of the right-justification normally used for numeric quantities, put a '-' in front of the number. E.g., "%-10s" will print the string starting in the first space and will pad with blanks until its total length is 10.
- The **calculateSavings()** method will require you to call each tax table's **caclulateTax()** method. However, the most useful version of this method requires you to pass in a **Couple** object. In this case you want to use the couple that calculateSavings() was invoked on. Recall, the **this** reference denotes the current object. Thus, you can use **this** as the actual parameter in your invocation.

Comments:
At a minimum provide a Javadoc comment explaining what each method does, and another one that summarizes each class. Include additional comments for lines that are especially complicated. At the top of the program include a comment with the following form:

```
/*
CSE 17
Your name
Your Lehigh e-mail
[if you used a tutor, provide his/her contact information here]
Program #1      DEADLINE: September 21, 2018
Program Description: Tax Tables
*/
```

Submission:
Once the program compiles, runs, and has been tested to your satisfaction, upload all three .java files to Course Site. To do so, click on the name of the assignment in the Course Site page, and then press the "Add submission" button at the bottom of the next page. Drag and drop each file into the area under "File submissions". If necessary, you can update your submission at any time before the deadline passes. Note, if you named your files anything other than **Couple.java**, **Person.java** and **TaxTable.java,** or they do not

contain the **Couple**, **Person,** and **TaxTable** classes, then you will lose points. Recall, your main method should be in the **TaxTable** class.

Sample Run:
A sample run is given below. User input is given in bold face:

```
> run TaxTable
2017 Married Filing Separately     2017 Married Filing Jointly
-------------------------------    -------------------------------
$0 - $9325:           10.0%        $0 - $18650:          10.0%
$9326 - $37950:       15.0%        $18651 - $75900:      15.0%
$37951 - $76550:      25.0%        $75901 - $153100:     25.0%
$76551 - $116675:     28.0%        $153101 - $233350:    28.0%
$116676 - $208350:    33.0%        $233351 - $416700:    33.0%
$208351 - $235350:    35.0%        $416701 - $470700:    35.0%
$235351+:             39.6%        $470701+:             39.6%

2018 Married Filing Separately     2018 Married Filing Jointly
-------------------------------    -------------------------------
$0 - $9525:           10.0%        $0 - $19050:          10.0%
$9526 - $38700:       12.0%        $19051 - $77400:      12.0%
$38701 - $82500:      22.0%        $77401 - $165000:     22.0%
$82501 - $157500:     24.0%        $165001 - $315000:    24.0%
$157501 - $200000:    32.0%        $315001 - $400000:    32.0%
$200001 - $300000:    35.0%        $400001 - $600000:    35.0%
$300001+:             37.0%        $600001+:             37.0%

The taxes Michelle and Joe owe under each tax table:
2017 Married Filing Separately      $11522.50
2017 Married Filing Jointly         $10317.50
2018 Married Filing Separately      $ 9749.00
2018 Married Filing Jointly         $ 8619.00

Savings for 2018 joint filers vs. 2017 joint filers:
Gary & Lisa: $21000000 / $50000     $ 553151.80
Henry & Ray: $140000 / $90000       $   7505.50
Michelle & Joe: $50000 / $25000     $   1698.50
Bob & Theresa: $20000 / $0          $     48.50
>
```

Note:
Although we use actual U.S. tax brackets and tax rates for both 2017 and 2018 in this assignment, the taxes owed computation is unlikely to be accurate in real life. This is because this assignment ignores exemptions and deductions. Furthermore, since the exemptions and deductions are different in both years, one should not use the results of this program to make statements about the relative merits of these tax codes.