Lab Partner 1 Name

Lab Partner 2 Name

CS 115 Fall 2019 Lab #6

Due: **Thursday, October 24th, midnight**

Points: **20**

Instructions:

1. Use this document template to report your answers. Enter all lab partner names at the top of first page.

2. You don't need to finish your lab work during the corresponding lab session.

3. ZIP your lab report and java files (if any) into a single ZIP file. Name the ZIP file as follows:

```
LastName_FirstName_CS115_Lab6_Report.zip
```

4. Submit the final document to Blackboard Assignments section before the due date. No late submissions will be accepted.

5. ALL lab partners need to submit a report, even if it is the same document.

Objectives:

1. (8 points) Write and test a user-defined class.

2. (12 points) Design a class for an everyday object, including required data and methods.

Problem 1 [8 points]:

Tasks:

Write and test a user-defined class (requiring conditions).

Write an application (client) program that uses an instance(s) of a user-defined class.

The federal income tax that a person pays is a function of the person's taxable income. The following table contains formulas for computing a single person's tax.

Bracket	Taxable Income	Tax Paid
1	$22,100 or less	15%
2	More than $22,100 but $53,500 or less	$3,315 plus 28% of the taxable income over $22,100
3	More than $53,500 but $115,000 or less	$12,107 plus 31% of the taxable income over $53,500
4	More than $115,000 but $250,000 or less	$31,172 plus 36% of the taxable income over $115,000
5	Over $250,000	$79,772 plus 39.6% of the taxable income over $250,000

Create a `FederalTax` class with the following:

- Declaration of an instance variable for the taxable income, a real number
- Declaration of constants for all the tax bracket income levels, the tax paid base amounts for each bracket, and the percents for each bracket
- Declaration of a constant `NumberFormat` to format all dollars
- Constructors - default (zero for the taxable income) & non-default
- Accessors - returns value of the instance variable
- Mutators - assigns new value to the instance variable, verify that the taxable income is non-negative, or assign zero
- a "public double taxPaid()" method that uses the above table to compute the tax
- a "public String toString()" method that displays the value of the instance variable AND the taxPaid.

Code then test (complete and check against Expected Result below) your methods by creating application (client) class FederalTaxApp.java to test your `FederalTax` class.

Complete the test plan below.

Implement your pseudocode in java. Be sure your program is appropriately documented. Accept user input from the keyboard.

Compile and run your program to see if it runs (no run-time errors).

Test your program with the test plan below. If you discover mistakes in your program, correct them and execute the test plan again.

Test plan			
Test case	Sample input data	Expected result	Verified
Tax Bracket 1			
Tax Bracket 2			
Tax Bracket 3			
Tax Bracket 4			
Tax Bracket 5			

Here is some sample output for a few of the test cases above, you must test them all.

Sample output:

```
Default FederalTax Object
Taxable Income: $0.00 Tax Paid: $0.00

Enter your taxable income: 52000
Updated FederalTax Object
Taxable Income: $52,000.00 Tax Paid: $11,687.00

Enter another taxable income: 137000
Non-Default FederalTax Object
Taxable Income: $137,000.00 Tax Paid: $39,092.00

Test a negative Income
Taxable Income: $0.00 Tax Paid: $0.00
```

Sample solution:

```
import java.text.NumberFormat;
public class FederalTax {
```

```
	//static constants  REQUIRED
	private final double BRACKET_1_INCOME = 0.,
	                     BRACKET_2_INCOME = 22100.,
	                     BRACKET_3_INCOME = 53500.,
	                     BRACKET_4_INCOME = 115000.,
	                     BRACKET_5_INCOME = 250000.;
	private final double BRACKET_1_BASE = 0.,
	                     BRACKET_2_BASE = 3315.,
BRACKET_3_BASE = 12107.,
	                     BRACKET_4_BASE = 31172.,
	                     BRACKET_5_BASE = 79772.;
	private final double BRACKET_1_PERCENT = .15,
	                     BRACKET_2_PERCENT = .28,
	                     BRACKET_3_PERCENT = .31,
	                     BRACKET_4_PERCENT = .36,
	                     BRACKET_5_PERCENT = .396;

	private final NumberFormat DOLLAR_FORMAT = NumberFormat.getCurrencyInstance();

	//instance variables
	double income;

	//default constructor
	public FederalTax() {
		setIncome(0);
	}

	//non-default constructor
	public FederalTax(double newIncome) {
		setIncome(newIncome);
	}

	//accessor methods
	public double getIncome() {
		return income;
	}

	//mutator methods
	public void setIncome(double newIncome) {
```

```
		if (newIncome>=0)
			income = newIncome;
		else income=0;
	}

	public double taxPaid() {
		if (income > BRACKET_5_INCOME)
			return	BRACKET_5_BASE	+	BRACKET_5_PERCENT*(income-
BRACKET_5_INCOME);
		else if (income > BRACKET_4_INCOME)
			return	BRACKET_4_BASE	+	BRACKET_4_PERCENT*(income-
BRACKET_4_INCOME);
		else if (income > BRACKET_3_INCOME)
			return	BRACKET_3_BASE	+	BRACKET_3_PERCENT*(income-
BRACKET_3_INCOME);
		else if (income > BRACKET_2_INCOME)
			return	BRACKET_2_BASE	+	BRACKET_2_PERCENT*(income-
BRACKET_2_INCOME);
		else return BRACKET_1_BASE + BRACKET_1_PERCENT*(income-BRACKET_1_INCOME);
	}

	public String toString() {
		return "Taxable Income: " + DOLLAR_FORMAT.format(income) +
		      "  Tax Paid: " + DOLLAR_FORMAT.format(taxPaid());
	}
}



import java.util.Scanner;
public class FederalTaxApp {
	public static void main(String[] args) {
		//declare Scanner class and interest, principal vars
		Scanner input = new Scanner(System.in);
		double income;

		//instantiate a default object of the FederalTax class
		FederalTax value1 = new FederalTax();
		System.out.println("Default FederalTax Object");
		System.out.println(value1.toString()+ "\n");
```

```
            //query for income
            System.out.print("Enter your taxable income: ");
            income = input.nextDouble();

            //change object and output
            value1.setIncome(income);
            System.out.println("Updated FederalTax Object");
            System.out.println(value1.toString()+ "\n");

            //query for another income
            System.out.print("Enter another taxable income: ");
            income = input.nextDouble();

            //instantiate an object of the FederalTax class
            FederalTax value2 = new FederalTax(income);
            System.out.println("Non-Default FederalTax Object");
            System.out.println(value2.toString()+ "\n");

            value2.setIncome(-1.);
            System.out.println("Test a negative Income");
            System.out.println(value2.toString()+ "\n");
        }
}
```

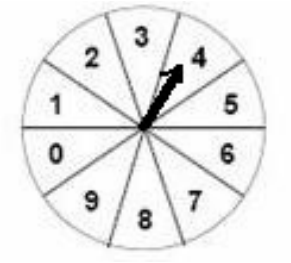
Problem 2 [12 points]:

Design a class for an everyday object, including required data and methods.

Define the private attributes (**and their data types and valid ranges**) and public methods (**and their arguments and return types**) for the following classes. **You do not need to code, just design**.

Populate provided tables (enter as many rows as you find necessary; add more if needed) with your answers. Feel free to add extra tables, boxes, comments, etc. if needed.

1. A fair 10-section game spinner (as shown below) **[2 points]**.



Private attributes (use "N/A", "undefined", "none", etc. If necessary)			
Attribute name	Data type	Valid range of values	Comments

Public methods (use "N/A", "undefined", "none", etc. If necessary)			
Method name	Return data type	Arguments	Comments

Sample solution:
private attributes: arrowNumber (integer 0 thru 9)

public methods:

default constructor(void, no arguments, initialize arrow at zero),

constructor(void, argument is initial arrow position in valid range),

getArrow (returns int, no arguments),

setArrow (void, integer argument is arrow position in valid range),

spin (void, no arguments, randomly position arrow),

toString (String, no arguments, return formatted string arrow position)

2. A fraction data type **[4 points]**.

Private attributes (use "N/A", "undefined", "none", etc. If necessary)			
Attribute name	Data type	Valid range of values	Comments

Public methods (use "N/A", "undefined", "none", etc. If necessary)			
Method name	Return data type	Arguments	Comments

Sample solution:

private attributes:

numerator (integer),

denominator (integer, >0). Assume a negative fraction stores the sign in the numerator

public methods:

default constructor(void, default fraction maybe 0/1),

constructor(void, 2 integer arguments numerator and denominator), setNumerator/setDenominator(void, integer argument, nonzero denominator, negative only allowed in numerator),

getNumerator/getDenominator (return int)

getDecimal (returns the decimal equivalent of the Fraction), add/substract/multiply/divide (returns Fraction, one argument Fraction, does the math and returns a new Fraction, zero denominator not allowed)

reduce (void, no args, reduce the fraction by removing common divisors)

3. A soda vending machine **[6 points]**.

Private attributes (use "N/A", "undefined", "none", etc. If necessary)			
Attribute name	Data type	Valid range of values	Comments

Public methods (use "N/A", "undefined", "none", etc. If necessary)			
Method name	Return data type	Arguments	Comments

Sample solution:

private attributes:

currentAmountDeposited (integer cents, zero or greater),

how many sodas of each type are available (array of integer counters, zero or greater),

how many coins for return of each type are available (dimes, nickels, quarters) (array of integer counters, zero or greater),

priceOfSoda (constant, static, integer, greater than zero)

public methods:

default constructor(void, empty SodaVendingMachine),

constructor(void, argument is array of how many sodas of each type are available, array of how many coins for return is available),

depositMoney (void, argument is cents deposited 5,10,25),

chooseSoda (returns soda, if enough money returns the soda as a String, argument is soda type chosen),

getChange (returns interger change amount, no argument)