GMU Spring 2023 – CS 211 – Project 2

Due Date: Friday, March 10th, 11:59pm

Changelog

- Typos in **`Taxation`**: the arrays are named **`taxRate`** and **`incomeBracket`**, not **`taxRates`** and **`incomeBrackets`**
- Clarification of the parameters of methods **`bracketIncome`** and **`bracketTaxRate`**
- A sample code is provided at the end of the document
- In the **`Family`** constructor, the first parameter is a number that corresponds to **`numMembers`**
- A flowchart of the taxation process was added
- You don't need to validate the number of decimal places inside **`setGrossIncome()`**
- In **`Person`** the ids must start from 1

Description

The purpose of this assignment is to practice basic object oriented programming principles with Java.

Instructions

- Honor Code: The project is **individual work** of each student, you're not allowed to collaborate in any form. Copying code from other sources (colleagues, websites, etc.) is a serious violation of the University's Honor Code. Your code will be examined for similarities with other sources.
- Validation: All method parameters must be properly validated (e.g. *income* can't be negative, *family members* can't be zero, etc.)
- Documentation: Comments in JavaDoc-style are required. You must comment each class, each method and each field, as well as any piece of code that it's not obvious what it does.
- Visibility: Class fields should not in general be visible/accessible to other classes. Create getter and setter methods for any attributes that you want to exchange with other objects.
- Addons: You may add your own helper methods or data fields. Additional methods can have any visibility but additional attributes must be private only.
- Packages: You may not import any package (or use the fully-qualified name of a class to bypass this restriction) except the following ones: `Scanner`, `File`,
- Restrictions: You are **not** allowed to use any language construct that hasn't been covered in class yet.

Submission

Submission instructions are as follows:

1. Upload all the source files (`*.java`) to Gradescope using the following link. Do **not** zip the files!

   https://www.gradescope.com/courses/498183/assignments/2703274

2. Download the files you just uploaded to Gradescope and compile/run them to verify that the upload was correct

3. Make a backup of your files on OneDrive using your GMU account

   If you skip steps 2 and 3 and your submission is missing the proper files, there won't be a way to verify your work and you will get zero points.

Assumptions

You may assume that:

- A child can have an income but this income cannot be a wage and, therefore, it's not subject to social security and medicare taxes or tax withholding.
- A married person cannot file as "single"
- A single-parent family will file as "single"
- We're not considering cases of adult dependents, only children can be dependents
- A couple has the option to file either "jointly" or "separately". When a couple is filing "separately" only one spouse is added to the family's tax return.

Grading

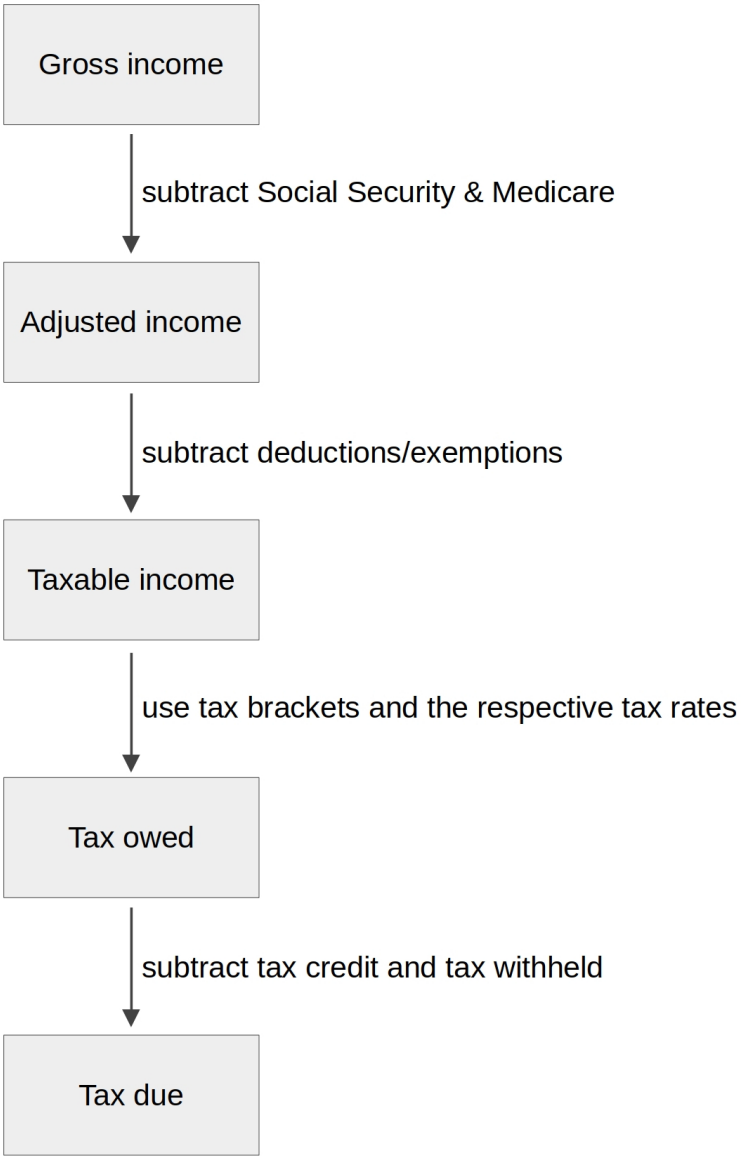
- Grading will be primarily automated.
- Manual grading will be used only for checking hard-coding, violations, comments, etc.
- When you upload your code to Gradescope, the built-in script will run some basic compliance checks for validation purposes only. **These are not logic tests**. The actual grading will take place afterwards and you can't see the tests that will be used.
- If your code doesn't pass all the compliance checks in Gradescope, it means that something fundamental is wrong and we won't be able to grade it unless you fix it. You will get **zero points** if you don't fix and reupload your code.

Testing

We won't provide a tester. The goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only. As a programmer, you must learn how to test your code on your own and you must also develop the ability to discover the corner cases that need to be tested.

Tasks

The tax season has started and the due date for filing your taxes is fast approaching (April 18). Your task in this project is to implement a simple calculator for your personal federal income tax. But you won't implement the complete logic of the federal taxation, just the basic calculations we encounter on the Form 1040. Below is a flowchart that provides an overview of the tax calculation process.



Disclaimer: For training reasons, the logic you're asked to implement here does not comply with all the IRS rules. Therefore, online resources may be misleading in this case, you're better off following the specs provided in this document.

You must implement the following classes based on the specification provided. Be reminded that this is the minimum implementation required, you're free to add more methods and attributes as you see fit. Added methods can have any visibility but added attributes must be private.

Person

id	Unique natural number (the type is your call)
name	**string** (cannot include any character other than a-z or A-Z or space)
birthday	**string** (in **YYYY/MM/DD** format)
ssn	**string** (nine-digit SSN in **xxx-xx-xxxx** format)
grossIncome	The gross income of a person (non negative **float** with two decimal places)
Person()	Constructor has no parameters. All it does is to initialize the **id**. Each object must have a unique **id** and all **id**s must be consecutive numbers starting from 1.
setName(string)	Returns **false** if invalid input or **true** otherwise
setBirthday(string)	Returns **false** if invalid input or **true** otherwise. Do not make a complete validation of a date, just check if the substrings between the **/** character contain digits only.
setSSN(string)	Returns **false** if invalid input or **true** otherwise
setGrossIncome(float)	Returns **false** if invalid input or **true** otherwise. No need to validate the number of decimal places, just the non-negative.
getGrossIncome()	getter method for **grossIncome**
getId()	getter method for **id**
toString()	Returns the person's name followed by masked SSN and birthday in the following format: **George Mason xxx-xx-1234 YYYY/**/****
deduction(Family)	Empty placeholder to be overridden by child classes. Returns **0.0**

Adult // inherits from Person

employer	**string** (without any restrictions)
Adult(name, birthday, ssn, grossIncome, employer)	Constructor with five parameters. It uses parent methods (when available) to set the respective attributes.
toString()	Overrides parent method. Returns the person's name followed by masked SSN, masked birthday, and gross income, in the following format: **George Mason xxx-xx-1234 YYYY/**/** $123456.78**
adjustedIncome()	• Returns the adjusted income of an employed adult. Adjusted income is what remains if you subtract the social security and the medicare taxes from the gross income. • The social security tax is calculated as a percentage of the gross income by using the **socialSecurityRate** • Gross income in excess of **socialSecurityIncomeLimit** is not subject to social security tax • The medicare tax is calculated as a percentage of the gross income by using the **medicareRate** • Unlike social security tax, medicare tax has no limit • The social security and the medicare taxes are split equally between employer and employee (i.e. the employee pays half of the calculated amount and the other half is paid by the employer). • You must not hard code the three parameters mentioned above. Instead, use the **Taxation** object to store and retrieve them.

taxWithheld()	Returns the total amount of tax that the Adult's employer withheld from the paychecks. Tax withholding is done at a progressive rate on the gross income. It's **10%** for the first $50K (inclusive), **15%** on the next $100K (inclusive), and **20%** thereafter.
deduction(Family)	• Overrides parent method • Returns the amount of the adjusted income that is exempted from taxation • The base exemption is **adultBaseExemption**. But if it's a single-parent **Family**, the base exemption is doubled. • If the adjusted income is over $100,000 the exemption is reduced by 0.5% per whole thousand dollars of adjusted income above 100K (i.e. if adjusted income is $113,478 the base exemption is reduced by 13%). Regardless of the adjusted income level, though, this reduction cannot exceed 30%. • Exemption can't be higher than the adjusted income
getEmployer()	getter method for **employer**

Child // inherits from Person

school	**string** (without any restrictions)
tuition	amount of tuition paid per year
Child(name, birthday, ssn, grossIncome, school, tuition)	Constructor with six parameters. It uses parent methods (when available) to set the respective attributes.
toString()	Overrides parent method. Returns the child's name followed by masked SSN, masked birthday, and the school, in the following format: **George Mason xxx-xx-1234 YYYY/**/** Fairfax High School**
getTuition()	getter method for **tuition**
deduction(Family)	• Overrides parent method • Returns the amount of the child's gross income that is exempted from taxation • The exemption is **childBaseExemption**. But if the child's **Family** has more than two children, each child's exemption is reduced by 5% for each additional child the **Family** has (i.e. for 3 kids is minus 5%, for 4 kids is minus 10%, and so forth). Regardless of the number of children, though, a child's exemption cannot be reduced by more than 50%. • Exemption can't be higher than the gross income

Family

numMembers	total number of family members (won't be higher than 50)
filingStatus	1 for single, 2 for married filing jointly, 3 for married filing separately
Person[] members	An array of family members
Family(numMembers, filingStatus)	Constructor initializes the two attributes and allocates memory for the array of members.
addMember(Person)	Adds a family member to the array

`getNumAdults()`	Returns the number of family members that are adults (i.e. parents)
`getNumChildren()`	Returns the number of family members that are children
`getFilingStatus()`	getter method for `filingStatus`
`getTaxableIncome()`	Returns the family's total taxable income. That is, the adjusted income of all the adults plus the gross income of all the children minus the respective deductions.
`taxCredit()`	• Returns the amount of tax credit that a family is eligible to receive • A family is eligible for a tax credit if its taxable income is in the low 50% of the `medianIncomePerCapita` (i.e. no more than half of the median income per capita) • The credit is $30 per each whole thousand dollars of taxable income • Each child is eligible for an additional tax credit which is equal to its tuition or $1000, whichever is lower. • If parents are filing separately, they can each claim half of the tax credit only. • Maximum credit per family is $2,000 or the amount of pre-credit tax, whichever is lower.
`calculateTax()`	• Returns the amount of tax that a family either owes or is to be refunded with. • This method must call the other methods and then use their results; it must not repeat any of the calculations that have already been done in the other methods. • Follow these steps in the order provided: 1. Calculate the taxable income 2. Find the maximum tax bracket that the taxable income corresponds to 3. Sum the amounts of tax that correspond to each tax bracket up to the maximum tax bracket. This is the *total calculated tax* 4. Calculate the tax credit and subtract it from the *total calculated tax* 5. Finally, subtract any taxes that were withheld during payroll

Taxation

`socialSecurityRate`	`float` value. Default value is 12.4 (for 12.4%)
`socialSecurityIncomeLimit`	`float` value. Default value is 137700.0 (for $137,700)
`medicareRate`	`float` value. Default value is 2.9 (for 2.9%)
`adultBaseExemption`	`float` value. Default value is 3000.0 (for $3,000)
`childBaseExemption`	`float` value. Default value is 2000.0 (for $2,000)
`medianIncomePerCapita`	`float` value. Default value is 31099.0 (for $31,099)

`loadParameters(String filename)`	Loads the above six variables from a text file. The file is guaranteed to have 6 lines only. Each line stores one of the six variables as a pair of name value. The variables can be stored in any order, not necessarily the one shown above. An example of the text file is the following:

```
medicareRate 2.9
medianIncomePerCapita 31099.0
socialSecurityIncomeLimit 137700.0
adultBaseExemption 3000.0
childBaseExemption 2000.0
socialSecurityRate 12.4
```

`incomeBracket` You can store this data any way you want but it has to be **`private`**. Other objects will use **`bracketIncome()`** to query this table.

bracket #	single	married (jointly)	married (separately)
1	$0 to $10,000	$0 to $20,000	$0 to $12,000
2	$10,000.01 to $40,000	$20,000.01 to $70,000	$12,000.01 to $44,000
3	$40,000.01 to $80,000	$70,000.01 to $160,000	$44,000.01 to $88,000
4	$80,000.01 to $160,000	$160,000.01 to $310,000	$88,000.01 to $170,000
5	$160,000.01 or more	$310,000.01 or more	$170,000.01 or more

`taxRate` You can store this data any way you want but it has to be **`private`**. Other objects will use **`bracketTaxRate()`** to query this table.

bracket #	single	married (jointly)	married (separately)
1	10%	10%	10%
2	12%	12%	12%
3	22%	23%	24%
4	24%	25%	26%
5	32%	33%	35%

`getNumTaxBrackets()` Returns the number of tax brackets. You need this because the above tables are private.

`maxIncomeTaxBracket(Family)` returns the maximum tax bracket that a family's income is in

`bracketIncome(Family, some_integer_type b)` Returns the portion of a family's taxable income that falls within bracket **`b`**. Bracket numbering starts from 1.

`bracketTaxRate(some_integer_type b, some_integer_type f)` Returns the tax rate that corresponds to bracket **`b`** and filing status **`f`**. Bracket numbering starts from 1.

TaxYear

`TaxYear(int max)`	The constructor takes one parameter, the maximum number of returns that can be filed for this year
`taxFiling(Family)`	Files a family's tax return for the year and validates the submitted data. If there is any kind of error (e.g. filing status doesn't agree with family members, a family doesn't have a parent, etc.), it doesn't accept the filing and returns **`false`**. If the data is valid it adds the Family to a local storage (the datatype is your call) and returns true. Families are stored in consecutive locations in the order they're added.
`taxWithheld()`	Returns the total tax that was withheld from all families' paychecks up to the moment of invoking this method (i.e. it may be called before tax filing for the year is complete)
`taxOwed()`	Returns the total tax that is owed by all families (i.e. based on their taxable income only), at the moment of invoking this method (i.e. it may be called before tax filing for the year is complete)
`taxDue()`	Returns the total tax that was due/returned the moment of invoking this method (i.e. it may be called before tax filing for the year is complete)
`taxCredits()`	Returns the total tax credits that were given to families the moment of invoking this method (i.e. it may be called before tax filing for the year is complete)
`numberOfReturnsFiled()`	Returns the number of tax returns that were filed
`numberOfPersonsFiled()`	Returns the total number of persons that are included in the tax returns that were filed.
`getTaxReturn(int index)`	Returns the **`Family`** that is stored at location **`index`** of the local storage.

Analytics

`povertyThreshold`	Default value for 2023 is $27,750 per family according to Census Bureau
`Analytics(TaxYear)`	Constructs an object that will be used for running various statistics on a certain tax year
`setPovertyThreshold(float)`	Setter method for **`povertyThreshold`**
`averageFamilyIncome()`	Returns the average taxable income per family for the given year
`averagePersonIncome()`	Returns the average taxable income per person for the given year
`maxFamilyIncome()`	Returns the maximum family taxable income for the given year
`maxPersonIncome()`	Returns the maximum personal taxable income for the given year
`familiesBelowPovertyLimit()`	Returns the number of families that have a taxable income below the poverty threshold
`familyRank(Family)`	Returns the rank of a family's taxable income in a certain tax year. Assume no other family has the exact same income. The rank of the highest income is 1.

Sample code for testing

```
Adult a1 = new Adult("name1", "1232/02/22", "987-65-4320", 0.00f, "GMU");
Adult a2 = new Adult("name2", "1332/02/22", "987-65-4321", 1234.56f, "GMU");
Adult a3 = new Adult("name3", "1432/02/22", "987-65-4322", 13456.78f, "GMU");
Adult a4 = new Adult("name4", "1572/02/22", "987-65-4323", 23979.54f, "GMU");
Adult a5 = new Adult("name5", "1632/02/22", "987-65-4324", 67890.12f, "GMU");
Adult a6 = new Adult("name6", "1732/02/22", "987-65-4325", 123456.78f, "GMU");
Adult a7 = new Adult("name7", "1876/05/01", "789-56-1236", 145000.98f, "Mason");
Adult a8 = new Adult("name8", "1932/02/22", "987-65-4327", 267890.12f, "GMU");
Adult a9 = new Adult("name9", "2032/02/22", "987-65-4328", 312346.78f, "GMU");
Child c1 = new Child("kid1", "1200/01/01", "999-65-1110", 0.0f, "FHS", 3300.0f);
Child c2 = new Child("kid2", "1300/01/01", "999-65-1111", 100.0f, "FHS", 0.0f);
Child c3 = new Child("kid3", "1400/01/01", "999-65-1112", 300.0f, "FHS", 0.0f);
Child c4 = new Child("kid4", "1500/01/01", "999-65-1113", 900.0f, "FHS", 900.0f);
Child c5 = new Child("kid5", "1600/01/01", "999-65-1114", 1600.0f, "FHS", 1234.0f);
Child c6 = new Child("kid6", "1700/01/01", "999-65-1115", 7300.0f, "FHS", 6650.0f);
Child c7 = new Child("kid7", "1800/01/01", "999-65-1116", 12000.0f, "FHS", 11999.0f);
Child c8 = new Child("kid8", "1900/01/01", "999-65-1117", 27000.0f, "FHS", 100.0f);
Child c9 = new Child("kid9", "2000/01/01", "999-65-1118", 41560.0f, "FHS", 8765.0f);
Family f1 = new Family((byte)2, (byte)3);
f1.addMember(a1);
f1.addMember(c1);
f1.calculateTax();
Family f2 = new Family((byte)4, (byte)2);
f2.addMember(a2);
f2.addMember(a3);
f2.addMember(c2);
f2.addMember(c3);
f2.calculateTax();
Family f3 = new Family((byte)3, (byte)2);
f3.addMember(a4);
f3.addMember(a5);
f3.addMember(c4);
f3.calculateTax();
Family f4 = new Family((byte)6, (byte)2);
f4.addMember(a6);
f4.addMember(a7);
f4.addMember(c5);
f4.addMember(c6);
f4.addMember(c7);
f4.addMember(c8);
f4.calculateTax();
Family f5 = new Family((byte)2, (byte)2);
f5.addMember(a8);
f5.addMember(a9);
f5.calculateTax();
Family f6 = new Family((byte)1, (byte)1);
f6.addMember(c9);
TaxYear y = new TaxYear(100);
y.taxFiling(f1);              // returns true
y.taxFiling(f2);              // returns true
y.taxFiling(f3);              // returns true
y.taxFiling(f4);              // returns true
y.taxFiling(f5);              // returns true
y.taxFiling(f6);              // returns false
y.numberOfReturnsFiled();     // returns 5
y.numberOfPersonsFiled();     // returns 17
y.taxWithheld();              // returns 142866.65
y.taxOwed();                  // returns 216255.72
y.taxDue();                   // returns 73119.06
y.taxCredits();               // returns 270.0
```