- Name: Carl Kaziboni
- Student ID: s2413182
- Tutorial group: 2
- Tutor: Andrey Demindenko
- Date: 2024-03-08

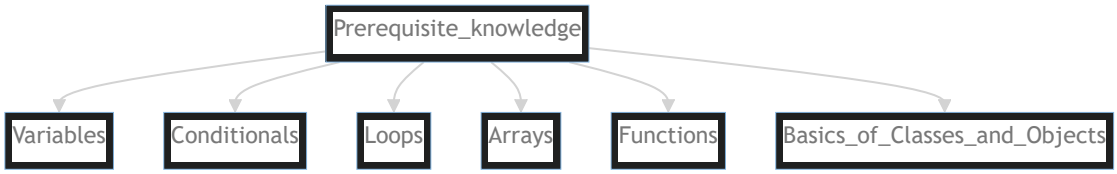
Calculate My Income Tax

Table of Contents

Target audience

Beginners in Object Oriented Programming and Java taking INF1B. They must have a few weeks of Object-Oriented Programming experience in Java.
Back to Table of Contents

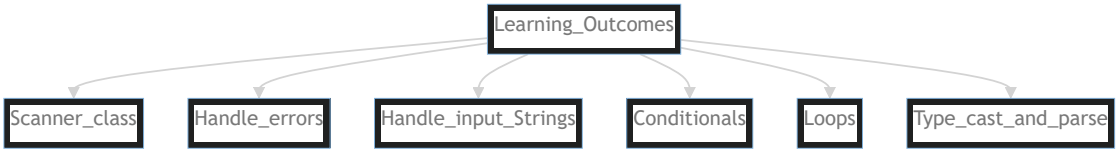
Prerequisite knowledge



- Variables
- Conditionals
- Loops
- Arrays
- Functions
- Basics of Classes and Objects

The elements in the list are needed specifically in Java. Enhanced for loops[6] are not needed for this assignment. Back to Table of Contents

Learning outcomes



On completion the student should be able to:

1. Use Scanner class and read from
2. Handle Errors in Java
3. Handle input Strings
4. Effectively use conditionals
5. Effectively use loops
6. Type cast[7] and parse[8]

Back to Table of Contents

Introduction

Ka-Ching! That is the sweet sound of the government cashing away part of your income. Have you always wondered how those pesky taxes are calculated. Do have a passion for programming? Well Calculate My Income Tax is perfect for you. Together we are going to level up your Java programming skills and all the while learning about legal robbery (oops I meant taxes). Throughout this worksheet we will learn to apply various skills you have learned and add a few more.

Back to Table of Contents

Algorithm

Let's get started! This section is divided into two parts, an easy to read description with a few Java technical terms (Don't worry we've provided a glossary for you to reference at any time, all underlined words are in the glossary) and a pseudocode section so you can really get a feel for the algorithm. There are also a few examples to

help you out.
Back to Table of Contents

Diagram

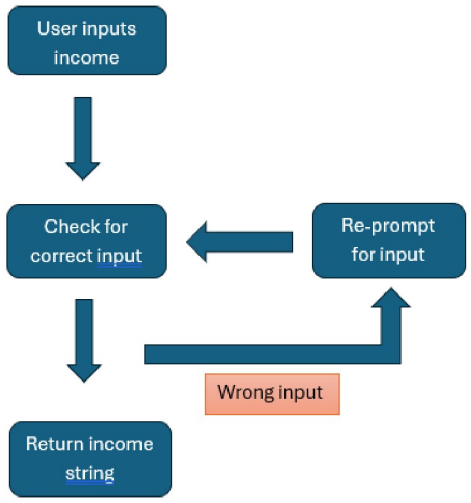


Figure 1. getIncome() flow chart

Input	Incorrect input / Correct input
Fifty thousand	Incorrect input
50000.00	Incorrect input
$50000	Incorrect input
50 000	Incorrect input
	Incorrect input
50000	Correct input

Figure 2. A table of acceptable/ non-acceptable input

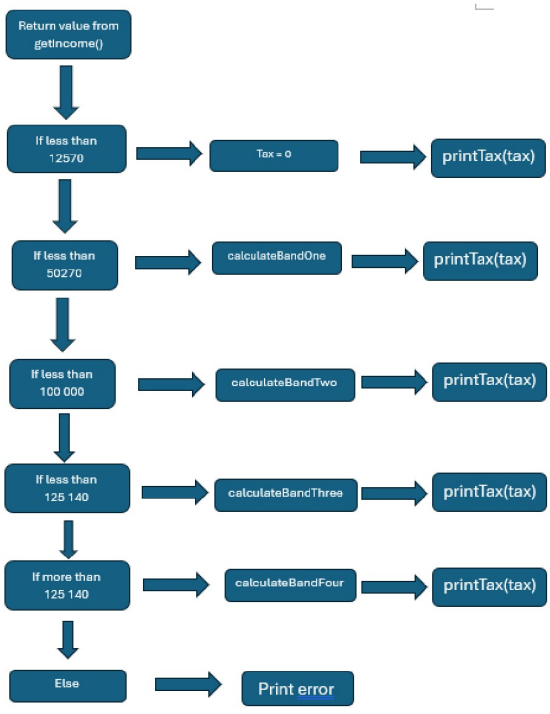


Figure 3. Flow chart for algorithm.

Input	Output
12570	0
50000	7486
80000	19432
120000	39432
200000	75588

Figure 4. Expect outputs for inputs.

Back to Table of Contents

- **Description**
  - Step 1: To get the user's income, otherwise how else would you calculate taxes. We're going to use via a special helper method[3] called `getIncome` located in our `GetNumerical` class. Firstly we need `import java.util.Scanner;` .In order to get input from the user we say `Scanner incomeInput = new Scanner(System.in);` . I know this seems like a lot but we're going to break it down together. You already know the basics of classes and objects, this is exactly the same however this time we specify `System.in` [5] in order to tell `Scanner` we are getting input from the console[2]. Easy, right?! Now onto the next step!
  - Step 2: We already know how get input from the user but how do we get the goodies of what's inside the input. We're going to use `nextLine()` to help us. `income = incomeInput.nextLine();` reads (this is how we talk about getting the goodies) the user input and assigns it to `income` .

```
System.out.print("Enter whole positive num
Scanner incomeInput = new Scanner(System.i
//read input obtained
String income = incomeInput.nextLine();
```

  - Step 3: Check whether the input is valid, I mean no one earns 'three dogs and cat' and we don't want input like that. For this task we're going to keep it simple and only accept positive whole

numbers from the user. We're going to use old friends of ours to help us achieve this. FOR and WHILE Loops! I know exciting right. So, we begin with a `while` loop and we're going to use variables `isEmpty` to check if our input is empty and `allNumerical` to check if input is all digits. Now we say `while (!isEmpty || !allNumerical)` to run the loop while both boolean variables are false. We will need to `import java.util.String;` .
To check if `income` is empty, ironically, we say `income.isEmpty()` . It's almost like coding in English. Now if `income` is empty we ask again for and input in the correct form. If `income` isn't empty we run our second check. Here comes in the `for` loop. We iterate over the string[4] and check if every character is a digit, and we do this with, you can guess it, another helped method[3]. We need to `import java.util.Character` . We check if every character is a digit by another almost English sounding method[3] called `isDigit` . To index a string[4] we use another helper function called `charAt` i.e `income.charAt[i]` . To check if a character is a digit we say `Character.isDigit(income.charAt(i))` , if the character is a digit we continue checking till `allNumerical` is True, if not we break from the loop and ask for the input again. Great we have checked obtained and checked our input and returned the input as a `String` ! (See Figure 1 and Figure 2).

```
//Keep prompting for input till input is c
      while(!notEmptyInput || !allNumerica
      {
          notEmptyInput = false;
          allNumericalInput = false;
          if (!income.isEmpty())
          {
              notEmptyInput = true;

              for (int i = 0; i < income.l
              {
                  if (!Character.isDigit(i
                  {
                      notEmptyInput = fals
                      break;
                  }
                  allNumericalInput = true
              }
          }
          //all constraints satisfied
          if (notEmptyInput && allNumerica
          {
              break;
          }
          System.out.print("Please enter i
          incomeInput = new Scanner(System
          income = incomeInput.nextLine();
      }
```

- Step 4: Now we turn our eyes to the main function, where all the magic happens. We take

our input from the user and assign it to `income`. However before we can make any calculations using we need to change the string[4] to an integer. To do this we parse[8] income to an integer using the `parseInt` after we `import java.util.Integer;`. However, sometimes there may be an error with what we give `parseInt`. When we have an error the method[3] will throw an exception, tell us the error but will crash the program. We need to handle these errors gracefully. We will use `try {} catch () {} finally {}`.

Between the `try` curly braces we will place in our `totalIncome = Integer.parseInt(income);`. This lets us try out some code first, see more English like terms. The error we specifically want to 'catch', see coding is definitely a sport, from our method[3] is called `NumberFormatException`, these are often shown in the Java Documentation. So we say `catch (NumberFormatException exception)` and between the curly braces we specify what we want to do. In our case we are going to give our user a nice note about what's wrong with their input and an example of what is the correct input. You probably already know about `System.out.printf` but in our case we will use a slightly different variant called `System.err.printf` to print out the error. And now we **gracefully** exit the program using `System.exit(1)`

In our `finally` section, this code will always be implemented, so between our curly braces we will just print out our income. Now we've handled

some errors, see everything is easier broken down into small steps.

```
int totalIncome = 0;
//Get input from user
String input = GetNumerical.getIncome();
//make sure input is integer
try
{
  totalIncome = Integer.parseInt(input);
}
catch (NumberFormatException exception)
{
  System.err.printf("%s not positive whole
  System.exit(1);
}
finally
{
  System.out.println("Income: " + totalInc
}
```

- Step 5: This is really where the fun begins, the moment we've been waiting for. IT'S TIME TO CALCULATE TAXES!! We know, no one is ever this excited for taxes but we are. First question we need to ask ourselves, what tax bracket are we? So we need to make conditionals of what tax bracket we're in, you know those `if` and `else if` things. Using if-else-if-else statements makes it easy to follow along and ensures we cover all edge cases. There are 5 cases, below 12570 (Personal Allowance), above 12570 and below 50270, above 50270 and below 100000, above 100000 and below 125140 and above

200000. We then have an `else` in case we have an unexpected input that passes all the checks but is not valid, this is all done in the id of robustness.
We're almost done calculating our tax, just a few small things. All the functions to calculate tax are found at the bottom of `Design` just to keep the code cleaner. They are named `calculateBandOne` to `calculateBandFour` (no case is need for below 12570 as `tax` is initialized to 0) and they apply the formulas (the specifics are not important) for calculating tax and all return an integer. However, we use 0.2 for example in our calculations which is a double, so we have what is called a type mismatch between what we return and calculated. To fix this we change our resulting `tax` to an integer by casting[7]. We type cast like this, `tax = (int) ((totalincome - PERSONAL_ALLOWANCE) * TAX_RATE_ONE);` , the `(int)` makes sure we assign an integer to tax.
All that's left to do is print out the tax using `printTax` which is a simple function that prints out 'Tax: [resulting income]' (See Figure 3 and Figure 4).

```
//start calculating tax
int tax = 0;
//Non-taxable income
if (totalIncome <= PERSONAL_ALLOWANCE)
{
    printTax(tax);
}
//First 37700 after personal allowance taxed a
```

```
else if (totalIncome <= (PERSONAL_ALLOWANCE +
{
    tax = calculateBandOne(totalIncome);
    printTax(tax);
}
//First 37700 after personal allowance taxed a
else if (totalIncome <= BAND_THREE)
{
    tax = calculateBandTwo(totalIncome);
    printTax(tax);
}
//First 37700 after personal allowance taxed t
else if (totalIncome <= BAND_FOUR)
{
    tax = calculateBandThree(totalIncome);
    printTax(tax);
}
//First 37700 taxed at 20% then next 99730 tax
else if (totalIncome > BAND_FOUR)
{
    tax = calculateBandFour(totalIncome);
    printTax(tax);
}
//If all checks were bypassed
else
{
    System.out.println("Inappropriate format f
}
```

- CONGRATULATIONS!!! You have calculated income tax, this algorithm should be helpful for all your adult working life time to come. We hope this description

helped you gain more confidence with your programming.

Back to Table of Contents

Pseudocode

1. Get user's income
2. While (invalid income input)
   Keep asking for valid income input
3. Parse income input to integer
   If (errors exist)
   Handle errors
   Terminate program
4. If (income < Personal Allowance)
   tax = 0
   printTax(tax)
   else if (income < (Personal Allowance + First tax band))
   tax = calculateBandOne(income)
   printTax(tax)
   else if (income < Third tax band) tax = calculateBandTwo(income)
   printTax(tax)
   else if (income < Fourth tax band)
   tax = calculateBandThree(income)
   printTax(tax)
   else if (income > Fourth tax band)
   tax = calculateBandFour(income)
   printTax(tax)
   else
   print(Input wrongly formatted)

Back to Table of Contents

Revision Questions

1. What class did we use to get user input? a: Scanner.
2. What were the two criteria we used for valid input? a: Not Empty, All digits.
3. What is the return type of `getIncome` ? a: String.
4. How do you Error handle? a: Try {} Catch() {} Finally {}.
5. What method did we use to Parse to an integer? a: parseInt().
6. How did we print an error?a: System.err.print().
7. How many cases did we have?a: 5.

Back to Table of Contents

Glossary

Term	Definition
Console	The terminal (the thing you see hackers typing into on movies)
Method	A function. It is a named block of code within a program or a class that performs a specific task
String	An array of characters
System.in	An input stream representing standard input
Enhanced for loops	A concise way to iterate over arrays or collections
Type casting	The process of converting one data type to another

Term	Definition
Parsing	Parsing is the process of converting data from one format to another, often involving the interpretation of a sequence of symbols

Back to Table of Contents

Further Reading

- Java Documentation
- Edinburgh University Inf1B

Back to Table of Contents

Original challenge question from CodeGolf

Calculate My Income Tax

Background

Here in the UK[1], these are the income tax rules:

- You get a personal allowance (untaxed) of up to £12,570:

    - If you earn less than £100,000, you get the full £12,570 as personal allowance
    - For every £2 over £100,000, your personal allowance goes down by £1
- After the personal allowance, the next £37,700 is taxed at the "basic rate" of 20%
- After that, the next £99,730 is taxed at the "higher rate" of 40%
- Finally, anything above this is taxed at the "additional rate" of 45%

1: This isn't actually the case in Scotland; only England, Wales and Northern Ireland.

Your task

Using the above tax rules, take in an annual salary (as a positive integer) and calculate the income tax.

Test cases

Input	Output
12570	0
50000	7486
80000	19432
120000	39432
200000	75588.5

Note: the final test case can be any of 75588, 75588.5, or 75589 (any is fine)

Clarifications

- You can choose whether to make the personal allowance an integer or keep it as a float
    - e.g. if the input is £100,003, the personal allowance can be £12,569, £12,568.50, or £12,568
- The same goes for the final output. If it ends up as a float, you can make it an integer or keep it as a float (see the final test case)
- This is code-golf, so shortest answer in bytes wins!