IT Academy 2018 Java stream

Java basics: flow control and arrays

Introduction

All development shall be done using IntelliJ IDEA. You must follow the good coding practices and format the code. Follow the instructions and make sure that all constraints are fulfilled. Good luck!

Project setup:

1. Open Intellij IDEA.
2. Create a new Java project called **`InterestCalculation`**.
3. Create a package named "`lt.swedbank.interestcalculator`".
4. Create a class called **`CompoundInterestCalculator`**. Class should be created within a specified package (see 3).
5. Create a **`main`** method within **`CompoundInterestCalculator`** class.

Requirements and general constraints:

- You are restricted to these data types: **`int`**, **`double`**, **`char`**, **`String`**. Any other primitive or complex types are not allowed!
- Input should be read from a console. Console should be used for output as well.
- Note: you do not need to implement input validation unless it is specified in task's conditions. Your application should work with the example input values only.

Compound interest

Compound interest is the addition of interest to the principal sum of a loan or deposit, or in other words, interest on interest. It is the result of reinvesting interest, rather than paying it out, so that interest in the next period is then earned on the principal sum plus previously accumulated interest. Compound interest is standard in finance and economics.[1]

$$P' = P\left(1 + \frac{r}{n}\right)^{nt}$$

where, P' – resulting amount, P – original (principal) amount, r – interest rate ([0; 1]), n – compounding frequency, t – overall length of time the interest is applied

[1] https://en.wikipedia.org/wiki/Compound_interest

Task 1. Calculate yearly compound interest:

- For this task, use compounding frequency = yearly (interest is re-calculated each year). This assumption simplifies the equation to this form:

$$P' = P(1 + r)^t$$

- Calculate compound interest for every year separately.
- **Input**: original (principal) amount, (annual) interest rate (%) and overall length of time the interest is applied (in years).
- **Output**: interest amount after each compounding period (year) and total amount (original amount + interest).

Example input:

```
 Amount: 100
Interest rate (%): 5
Period length (years): 3
```

Example output:

```
Interest amount after year 1: 5.00
Interest amount after year 2: 10.25
Interest amount after year 3: 15.76
Total amount: 115.76
```

Constraints:

1. You must use at least one *looping* statement. Which looping statement to use, it is up to you. Choose the most appropriate one for the task.
2. Format outputted amount(s) to 2 decimal places (hints: "**`System.out.printf(…, …)`**", "%.2f").

Task 2. Calculate compound interest using compounding frequency:

- User should be able to specify compounding frequency (how many times interest should be re-calculated during a year). For this task use the original compound interest equation:

$$P' = P\left(1 + \frac{r}{n}\right)^{nt}$$

- **Input**: compounding frequency code in addition to **Task 1** inputs.
- **Output**: interest amount after each compounding period (year) and total amount (original amount + interest) (same as for **Task 1**).

Example input:

```
      Amount: 100
      Interest rate (%): 5
      Period length (years): 3
      Compound frequency: M
```

Example output:

```
      Interest amount after year 1: 5.12
      Interest amount after year 2: 10.49
      Interest amount after year 3: 16.15
      Total amount: 116.15
```

Constraints:

1. Application should accept one of these compounding frequency codes:
    a. D – daily (365 times a year)
    b. W – weekly (52 times a year)
    c. M – monthly (12 times a year)
    d. Q – quarterly (4 times a year)
    e. H – half yearly (2 times a year)
    f. Y – yearly (1 time a year)
2. Compounding frequency code to compounding frequency (n) conversion should be done using a ***switch*** statement. If invalid code is entered, yearly compounding frequency should be used by default.

Task 3. Calculate all intermediate interest amounts:

- Calculate interest for every compounding iteration separately. For example: if monthly compounding frequency is used and interest is being calculated for 3 year period, calculation should be done in 36 (12 * 3) iterations.
- **Input**: same as for **Task 2**.
- **Output**: calculated intermediate interest amounts in addition to **Task 2** output.

Example input:

```
     Amount: 100
    Interest rate (%): 5
    Period length (years): 3
    Compound frequency: Y
```

Example output:

```
    Interest amount after year 1: 5.00
    Interest amount after year 2: 10.25
    Interest amount after year 3: 15.76
    Intermediate interest amounts: [5.0, 5.25, 5.512500000000017]
    Total amount: 115.76
```

Constraints:

1. Store calculated intermediate interest amounts in an *array*.
2. Array element should contain only interest amount calculated for a single compounding iteration.
3. Output array using a single Java statement.

Task 4: Generate intermediate interest amounts comparison table:

- For annual interest rates comparison purposes, generate intermediate interest amounts comparison table. This table should represent how much interest is calculated for each compounding iteration with distinct annual interest rates.
- **Input**: multiple annual interest rates (%) in addition to **Task 3** inputs.
- **Output**: intermediate interest amounts comparison table.

Example input:

```
 Amount: 100
Interest rate (%): 3
Interest rate (%): 5
Interest rate (%): 7
Interest rate (%): 0
Period length (years): 3
Compound frequency: Y
```

Example output:

```
3.00 3.09 3.18
5.00 5.25 5.51
7.00 7.49 8.01
```

Constraints:

1. Application should accept unlimited amount of interest rate values (amount of inputs is not predetermined in any way). Interest rates should be inputted one-by-one. Inputting should be terminated by "0" input. "0" input should not be used in calculations. Store interest rates into an array. ***do-while*** loop statement should be used for inputting interest rates.
2. Calculated intermediate interest amounts should be stored into a ***matrix***. You are free to choose which dimension represents iteration and which annual interest rates (in example above: X axis – iteration (time), Y – annual interest rates).
3. Use a nested for-each statement for printing comparison table.

Task 5: Validate user input:

- Validate user input according rules provided (see constraints below).
- **Input** same as for **Task 4**.
- **Output**: same as for **Task 4**.

Example input:

```
Amount: -1
Invalid input! Try again!
Amount: 0
Invalid input! Try again!
Amount: 100
Period length (years): 0
Invalid input! Try again!
Period length (years): 1.5
Invalid input! Try again!
Period length (years): 3
Compound frequency: A
Invalid input! Try again!
Compound frequency: Y
Interest rate (%): 3
Interest rate (%): -1
Invalid input! Try again!
Interest rate (%): 5
Interest rate (%): 100.01
Invalid input! Try again!
Interest rate (%): 7
Interest rate (%): 0
```

Example output:

```
3.00 3.09 3.18
5.00 5.25 5.51
7.00 7.49 8.01
```

Constraints:

1. If invalid input is detected, application user should be able to re-enter it for unlimited amount of times until input is considered valid.
2. Validation rules by input field:
    a. Amount – floating point number, greater than 0.
    b. Period length – integer, greater than 0.
    c. Compound frequency – character, allowed values: {'D', 'W', 'M', 'Q', 'H', 'Y'}.
    d. Interest rate – floating point number, from within the range (0; 100].