CMM024 Object Oriented Programming

Practical Session: 3

1. Write a method to compute the area of any circle and use this method to display the result

In the lecture there is an example on how to create your own method to compute the circumference of a circle. Use this a guide to create a method to return the are given a radius.

Note:

The formula for the area of a circle is given below

$$circle\ area = \pi r^2$$

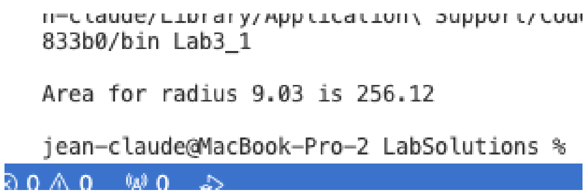
Where:

r is the radius of the circle.

Tasks:

1) Create a **Lab3_1.java** java file, and create a **main** function
2) Create code store the radius of a circle as a decimal number (not whole number).
3) Add code to assign a random value for the radius, which should be in the range of 5 to 10.
4) Code a method to return the area of any circle.
5) In the main method, invoke the method you created to display the area value of the random radius.
6) Run the program so the output is like the one shown below.

Output:



```
n-claude/Library/Application\ Support/Cod
833b0/bin Lab3_1

Area for radius 9.03 is 256.12

jean-claude@MacBook-Pro-2 LabSolutions %
```

2. Write a Dice game program (Part 1)

Brief:

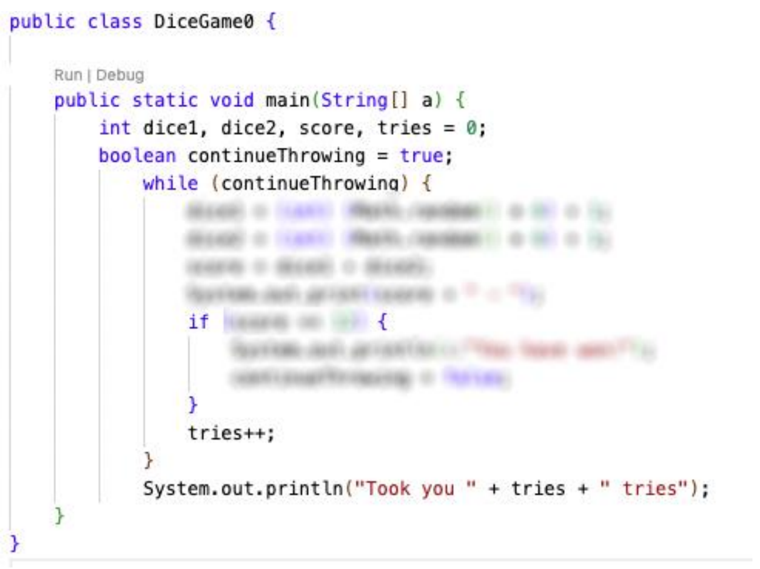
The game is about rolling two dices, and if the sum of the two dices is 12 i.e., two sixes, you win, otherwise keep on throwing the dice until you do.

Notes:

This is a typical situation where you must use a while loop.

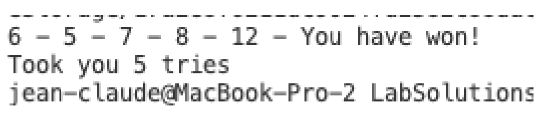
```
Number of tries = 0
Continue throwing = true
While (Continue throwing){
        Get first dice value
        Get second dice value
        Sum both dice values
        Display score value
        If(sum = 12)
                Display message you won
                Exit loop (Continue throwing = false)
        Increase number of tries
}
Display number of tries
```

Code setup



```
public class DiceGame0 {

    Run | Debug
    public static void main(String[] a) {
        int dice1, dice2, score, tries = 0;
        boolean continueThrowing = true;
            while (continueThrowing) {
                [illegible]
                [illegible]
                [illegible]
                [illegible]
                if [illegible] {
                    [illegible]
                    [illegible]
                }
                tries++;
            }
            System.out.println("Took you " + tries + " tries");
    }
}
```

Output:



```
6 - 5 - 7 - 8 - 12 - You have won!
Took you 5 tries
jean-claude@MacBook-Pro-2 LabSolutions
```

3. Write a Dice game program (Part 2)

Brief:

In the previous exercise, you have created a dice game, which runs only one time. Modify the code so the game runs 10 times and compute the total number of it took you to complete these 10 games.

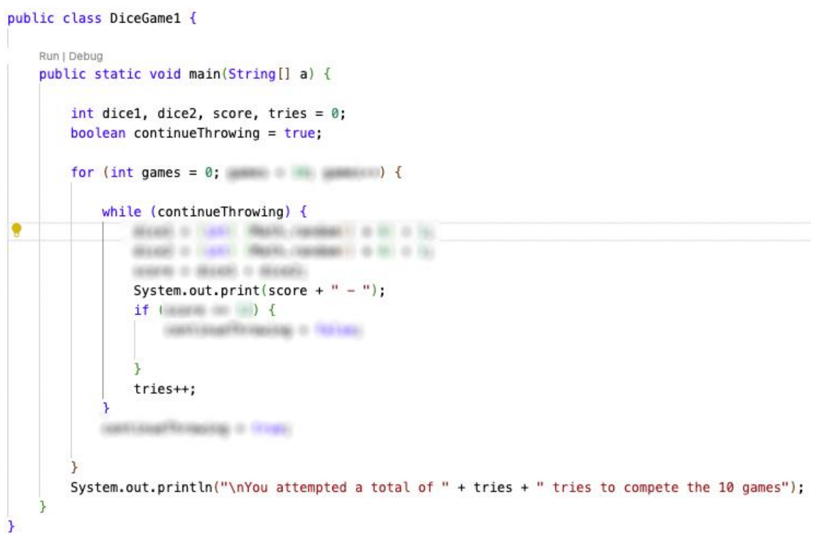
Notes:

This is a typical situation where you must nest a while loop into a for loop, where a few changes must be made so the 10 games run well.

Note: for each game, the boolean value controlling the while loop must be reset.

```
Number of tries = 0
Continue throwing = true
For each of the 10 games
        While (Continue throwing){
                Get first dice value
                Get second dice value
                Sum both dice values
                Display score value
                If(sum = 12)
                        Exit loop (Continue throwing = false)
                Increase number of tries
        }
        Reset while loop (Continue throwing = true)
}
Display number of overall tries
```

Code setup



```
public class DiceGame1 {

    Run | Debug
    public static void main(String[] a) {

        int dice1, dice2, score, tries = 0;
        boolean continueThrowing = true;

        for (int games = 0; [illegible]) {

            while (continueThrowing) {
                [illegible]
                [illegible]
                [illegible]
                System.out.print(score + " - ");
                if ([illegible]) {
                    [illegible]

                }
                tries++;
            }
            [illegible]

        }
        System.out.println("\nYou attempted a total of " + tries + " tries to compete the 10 games");
    }
}
```

4. Write a Dice game program (Part 3)

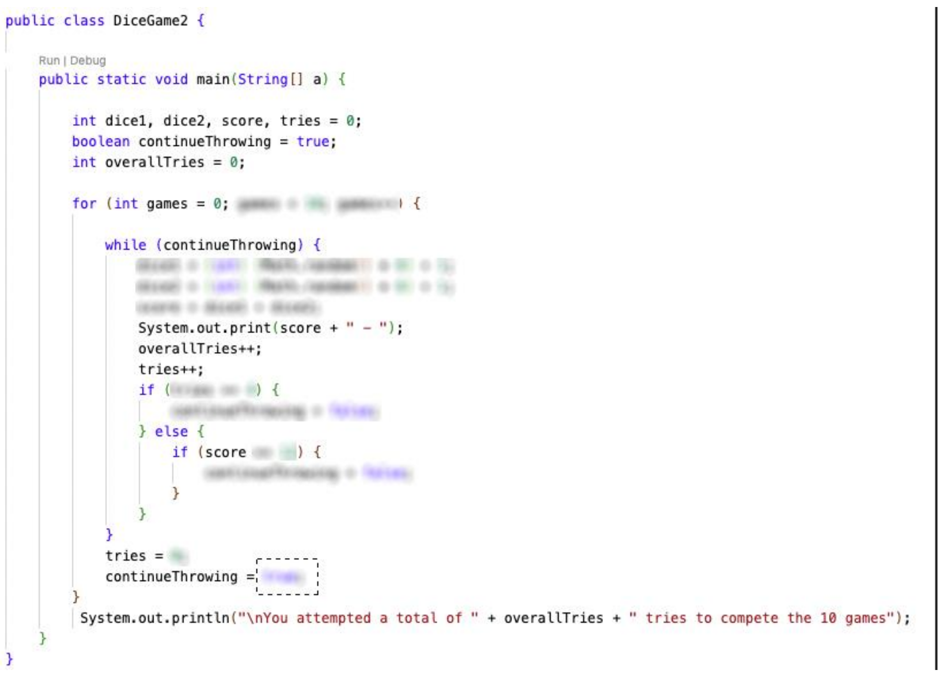
Brief:

In the previous exercise, you have created a dice game, which runs 10 times. Modify the code so the game still runs 10 times but add an extra rule which stipulate that you only have 4 attempts to win the game i.e., 4 attempts to get a double six. If you don't you lose., hence you are now more likely to lose!

This adds some complexity. For each game we need to track of the number of tries >= 4. This means that when a particular game completes, this value must be reset to 0. How do we track the overall number of attempts? We need to add another variable that will increase for all 10 games.

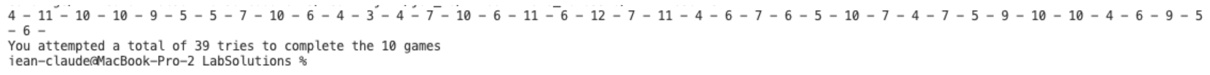
```
Number of tries = 0
Overall number of tries = 0;
Continue throwing = true
For each of the 10 games{
        While (Continue throwing){
                Get first dice value
                Get second dice value
                Sum both dice values
                Display score value
                Increase number of tries
                Increase overall number of tries
                If(number of tries >= 4)
                        Exit loop (Continue throwing = false)
                Else
                        If(sum = 12)
                                Exit loop (Continue throwing = false)
        }
        Reset while loop (Continue throwing = true)
        Reset number of tries (set it back to 0)
}
Display information
```

Code setup



```
public class DiceGame2 {

    Run | Debug
    public static void main(String[] a) {

        int dice1, dice2, score, tries = 0;
        boolean continueThrowing = true;
        int overallTries = 0;

        for (int games = 0; [redacted]) {

            while (continueThrowing) {
                [redacted]
                System.out.print(score + " - ");
                overallTries++;
                tries++;
                if ([redacted]) {
                    [redacted]
                } else {
                    if (score [redacted]) {
                        [redacted]
                    }
                }
            }
            tries = [redacted]
            continueThrowing = [redacted]
        }
        System.out.println("\nYou attempted a total of " + overallTries + " tries to compete the 10 games");
    }
}
```

Output:



```
4 - 11 - 10 - 10 - 9 - 5 - 5 - 7 - 10 - 6 - 4 - 3 - 4 - 7 - 10 - 6 - 11 - 6 - 12 - 7 - 11 - 4 - 6 - 7 - 6 - 5 - 10 - 7 - 4 - 7 - 5 - 9 - 10 - 10 - 4 - 6 - 9 - 5
- 6 -
You attempted a total of 39 tries to complete the 10 games
iean-claude@MacBook-Pro-2 LabSolutions %
```

5. Write a Dice game program (Part 4)

Brief:

In the previous exercise, you have added another constraint for losing i.e., only 4 attempts to throw a double 6. As you may have noticed it takes a lot less tries to complete the 10 games but you were more likely to lose due to this extra condition.

In this final exercise, you are going to increase the chances of winning by adding another constraint which is if you throw two dices with the same value i.e., 2 x 3, 2 x 4 etc, you are given an extra move.

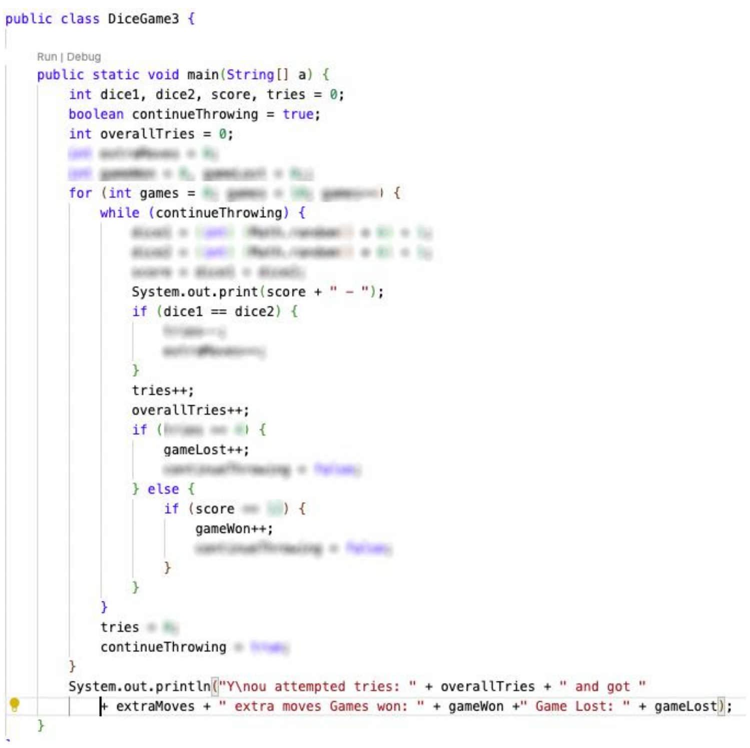
In addition, we are going track how many free moves are given, how many games were lost or won.

In terms of coding for this new rule, this is simply about decreasing the *tries* value, when the dice values are the same.

```
Number of tries = 0
Overall number of tries = 0;
Continue throwing = true
Extra moves number = 0
```

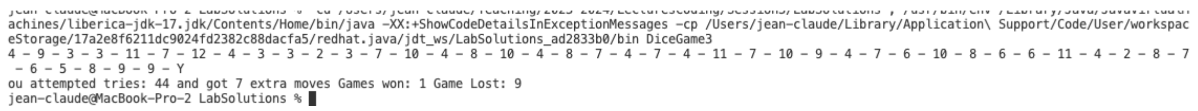
```
Game won number = 0
Game lost number = 0
For each of the 10 games
        While (Continue throwing){
                Get first dice value
                Get second dice value
                Sum both dice values
                Display score value
                If( Dice 1 value equal Dice 2 value)
                        Decrease number of tries
                        Increase Extra moves number
                Increase number of tries
                Increase overall number of tries
                If(number of tries >= 4)
                        Increase Game lost Number
                        Exit loop (Continue throwing = false)
                Else
                        If(sum = 12)
                                Increase Game Won Number
                                Exit loop (Continue throwing = false)
        }
        Reset while loop (Continue throwing = true)
        Reset number of tries (set it back to 0)
}
Display information
```

Code setup



```
public class DiceGame3 {

    Run | Debug
    public static void main(String[] a) {
        int dice1, dice2, score, tries = 0;
        boolean continueThrowing = true;
        int overallTries = 0;
        [redacted]
        for (int games = [redacted]) {
            while (continueThrowing) {
                [redacted]
                System.out.print(score + " - ");
                if (dice1 == dice2) {
                    [redacted]
                }
                tries++;
                overallTries++;
                if ([redacted]) {
                    gameLost++;
                    [redacted]
                } else {
                    if (score [redacted]) {
                        gameWon++;
                        [redacted]
                    }
                }
            }
            tries [redacted]
            continueThrowing [redacted]
        }
        System.out.println("Y\nou attempted tries: " + overallTries + " and got "
            + extraMoves + " extra moves Games won: " + gameWon +" Game Lost: " + gameLost);
    }
```

Output:



```
jean-claude@MacBook-Pro-2 LabSolutions % cd /Users/jean-claude/Teaching/2023-2024/LecturesCoding/Sessions/LabSolutions ; /usr/bin/env /Library/Java/JavaVirtualM
achines/liberica-jdk-17.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/jean-claude/Library/Application\ Support/Code/User/workspac
eStorage/17a2e8f6211dc9024fd2382c88dacfa5/redhat.java/jdt_ws/LabSolutions_ad2833b0/bin DiceGame3
4 - 9 - 3 - 3 - 11 - 7 - 12 - 4 - 3 - 3 - 2 - 3 - 7 - 10 - 4 - 8 - 10 - 4 - 8 - 7 - 4 - 7 - 4 - 11 - 7 - 10 - 9 - 4 - 7 - 6 - 10 - 8 - 6 - 6 - 11 - 4 - 2 - 8 - 7
 - 6 - 5 - 8 - 9 - 9 - Y
ou attempted tries: 44 and got 7 extra moves Games won: 1 Game Lost: 9
jean-claude@MacBook-Pro-2 LabSolutions % █
```

6. (Advanced) - Write a program that compute capital gain with compounded saving accounts.

Brief:

You are required to create a program that compute capital gain for a saving account. You start with an initial investment, and let it grow given an interest rate over several years. Bank often provides yearly compound interest rate in the range of 4 to 4.5% these days.

In this exercise, you are given a formula, which needs to be ported to code so you can perform the computation that is needed to solve the problem!

Test values:

	Value	**Variable name**
Initial Principle	100	principle
Yearly Interest Rate	4.5	yearlyInterestRate
Capital Gain	10th year	year

Note:

Below is an explanation and the formulas to code.

The rate increase is important. If you save some money in the bank account, and you are given an interest rate, the initial capital in your account will grow each year at a specific rate given the yearly interest rate you were given. However, we need to compute the increase ratio i.e., rateIncreaseRatio below.

$$rateIncreaseRatio = 1 + \left(\frac{yearlyInterestRate\ (\%)}{100}\right)$$

Where:

RateIncrease	Is the rate at which the saving will grow after each year. It is usually (1 + interest rate) with the interest rate as a decimal number. Hence if you

	have a yearly interest rate of 4.5%, this is 4.5/100 = 0.045 as a decimal. Since it is a yearly rate, and we need the monthly one for our computations as our interest is based on monthly payments. Therefore, we divide this rate by 12 (12 months in a year), giving us the RateIncrease equal to 1.00375.

$$rateIncreaseRatio = 1 + \left(\frac{4.5}{100 * 12}\right) = 1 + \left(\frac{4.5}{1200}\right) = 1.00375$$

Problem

There are many instances where you want to know how much your initial investment has grown after a specific year. The formula below computes this.

Use the Math.pow(…) method in the calculation below.

$$New\ Capital\ (Year) = \ principle * rateIncreaseRatio * \left(\frac{(rateIncreaseRatio^{Year} - 1)}{(rateIncreaseRatio - 1)}\right)$$

Where:

New Capital	If we want to find out what is the new capital after a certain year, let's say after the 10th year, then we need to compute the value as shown in the formula above
Principle	Is the initial amount in the bank account you deposited
RateIncrease	Discussed above. This is the rate at which your money grows after each payment
Year	A specific year i.e., the 10th year.

The Capital gain is simply the (new capital – the initial one). After 10 years, your initial capital of £100 will have grown to £1056.68 given a 4.5% interest rate. Some quick explanations are given in Appendix. Thus, the gain is 1056.68 – 100 = 956.68 over 10 years.

Tasks:

Part 1:

1. Create a file java called **investment1.java** and create the main function
2. Declare variables to hold the values given in the brief.
3. Declare (and name properly) a variable to hold the rate increase.

4. Create the code to compute the rate increase given above yearly rate in the brief and store the value.

5. Declare (and name properly) a variable to hold the new principle.

6. Create the code to compute the capital gain and store this value. For $(rateIncrease^{Year})$ use Math.Pow(…) function.

Part 2:

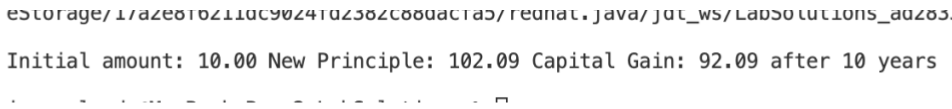
Rewrite the code and create two methods. One for the rateIncreaseRatio computation and one for the capital gain. The rate increase should only have one parameter and the method computing the capital gain, should have 3.

Part 3:

Use a for loop to compute the gain for each year until the year to check the capital gain and display this information in a proper manner.

<u>Output:</u>

Using the printf method.



```
eStorage/17a2e8f6211dc9024fd2382c88dacfa5/redhat.java/jdt_ws/LabSolutions_ad2833b0/

Initial amount: 10.00 New Principle: 102.09 Capital Gain: 92.09 after 10 years
```

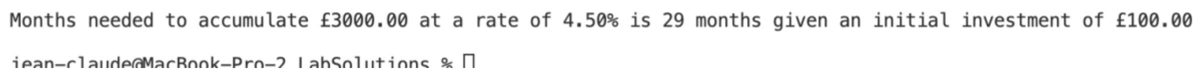
Part 4:

In this exercise, you want to discover how many months are needed to accumulate a specific capital. i.e., how many months before your account reaches 3000 (wantedPrinciple below) given an initial principle of 100. Given the information in the brief you would have to wait 29 months finally have this amount. This is what you are going to code.

Write a method to compute that value. Use the Mat.log(…) method. Make sure you have the right data type and number for the parameters. Then use this method to display that value. Use the Math.ceil(…) method to round up the value obtained for display. You need to cast this value as an integer as Ceil returns a double.

$$Months = \frac{log\left(\left(\frac{wantedPrinciple}{(principle * rateIncrease)/(rateIncrease - 1)}\right) + 1\right)}{log(rateIncrease)}$$

<u>Output:</u>



```
Months needed to accumulate £3000.00 at a rate of 4.50% is 29 months given an initial investment of £100.00

jean-claude@MacBook-Pro-2 LabSolutions % ▯
```

7. (Even More Advanced) - Write a program to compute various aspect of compounded loans

This is also a real example. When performing these sorts of computations, the most difficult is to position brackets. Below is a good example. The formulas below provides a clue.

<u>Test values:</u>

Inputs	**Value**	**Variable name**
Loan Amount	100000	loanAmount
Yearly Interest Rate	8	yearlyInterestRate
Years	25	years
Test Results	**Results**	
yearlyInterestRate	1.08	
monthlyPayment	780.66	

Notes for Part 1:

Rate at which the amount you owe will increase yearly.

$$\mathrm{yearlyIncreaseRatio} = 1 + \left(\frac{yearlyInterestRate\ (\%)}{100}\right)$$

To compute the yearly payment amount to cover the cost of the loan and to pay it off. This is the whole formula for this

$$yearlyPaymentAmount = \left(\frac{\mathrm{loanAmount} * \mathrm{yearlyIncreaseRatio}^{years}}{\left(\frac{yearlyIncreaseRatio^{years} - 1}{yearlyIncreaseRatio - 1}\right)}\right)$$

However, if it sometimes very useful to split this formula as shown below. This ensure we can code methods more easily and the become re-usable. Note *loanCostNoPayment* below will also be used in Part 2!

we first compute the cost of the loan without any payments for the final year (years). This can be coded in one method later.

$$loanCostNoPayment = \mathrm{loanAmount} * \mathrm{yearlyIncreaseRatio}^{years}$$

Then we can compute the yearly payment needed following the formula below.

$$yearlyPaymentAmount = \left(\frac{loanCostNoPayment}{\left(\frac{yearlyIncreaseRatio^{years} - 1}{yearlyIncreaseRatio - 1}\right)}\right)$$

To compute the monthly payment, you need to divide the yearly payment amount by 12.

$$montlyPayment = \frac{yearlyPayment}{12}$$

We can combine these formulas as shown below. This can be therefore code in one method later.

$$montlyPayment = \frac{\left(\frac{loanCostNoPayment}{\left(\frac{yearlyIncreaseRatio^{years} - 1}{yearlyIncreaseRatio - 1}\right)}\right)}{12}$$

Notes for Part2:

Now that we have the yearly increase ratio and the monthly payment, we can list interesting aspects of the loan over the years etc.

We need to perform some calculations first. The entire formula to calculate the amount a borrower owes at a specific year is:

$$amountOwed = loanAmount * yearlyIncreaseRatio^{year} - \left(\frac{(12 * monthlyPayment * (yearlyIncreaseRatio^{years} - 1))}{(yearlyIncreaseRatio - 1)}\right)$$

However, it is often easier to split complex formula into smaller ones, as this renders the programming a lot easier and will allow you to create a series of methods to perform these calculations.

The *getLoanCostNoPayment* is also needed in Part 1.

$$loanCostNoPayment(year) = loanAmount * yearlyIncreaseRatio^{year}$$

To calculate the deductions from the yearly loan cost due to payments at a specific year. Again, note that the *yearlyIncreaseRatio* is also needed in Part 1. Also, remember that you make 12 payments in a year.

$$deductionsRatio\ (year) = 12 * \left(\frac{yearlyIncreaseRatio^{year} - 1}{yearlyIncreaseRatio - 1}\right)$$

So, the amount owed is the total loan cost for a specific year minus the deductions (payment contributions) for that year.

$$amountOwed(year) = loanCostNoPayment - (deductionsRatio * monthlyPayment)$$

Tasks

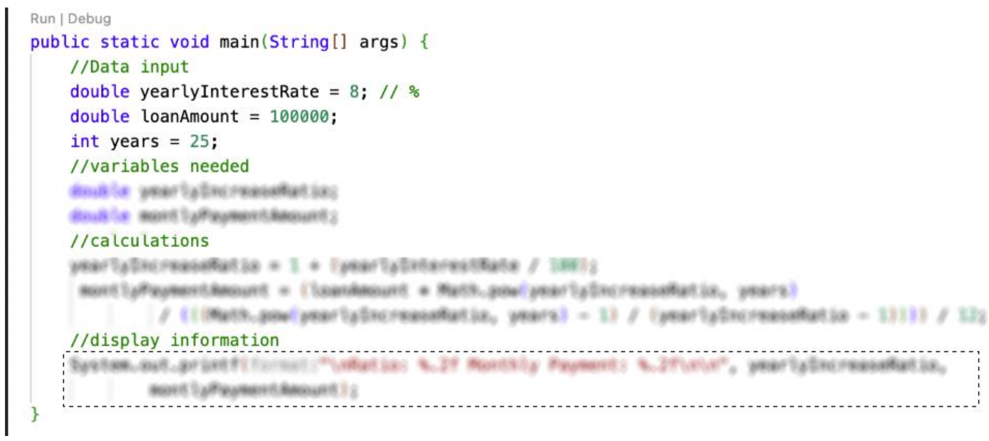
In Part 1, you should code the formulas in the main method first. You must declare all the necessary variables to hold the values. As a hint, suggestions of variable names have been used in these formulas and the test values table. Display the wanted values in the console.

Once the results you display in the console are as the same as the one given above, create three methods as shown in the screenshots below.

Re-run your code to ensure you obtain the same values.

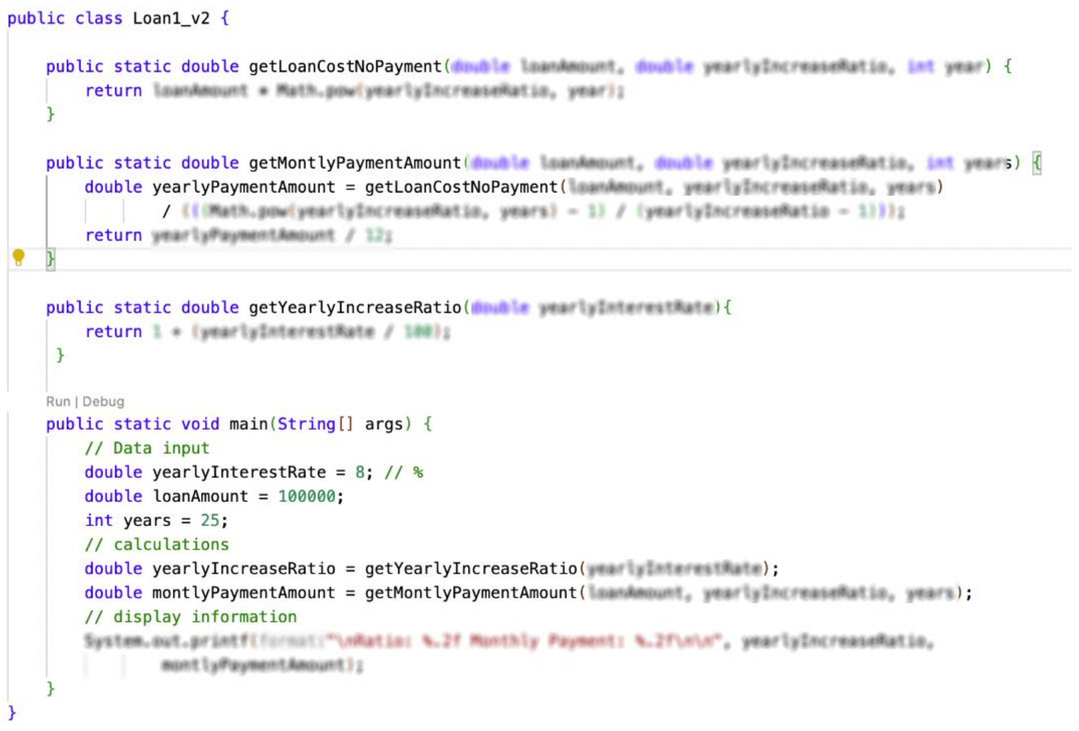
Code for Part1:

First:



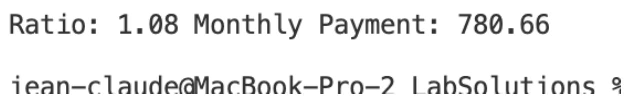
```
Run | Debug
public static void main(String[] args) {
    //Data input
    double yearlyInterestRate = 8; // %
    double loanAmount = 100000;
    int years = 25;
    //variables needed
    [redacted]
    //calculations
    [redacted]
    //display information
    [redacted]
}
```

Then:



```
public class Loan1_v2 {

    public static double getLoanCostNoPayment([redacted]) {
        return [redacted]
    }

    public static double getMontlyPaymentAmount([redacted]) {
        double yearlyPaymentAmount = getLoanCostNoPayment([redacted])
                / [redacted]
        return [redacted]
    }

    public static double getYearlyIncreaseRatio([redacted]){
        return [redacted]
     }

    Run | Debug
    public static void main(String[] args) {
        // Data input
        double yearlyInterestRate = 8; // %
        double loanAmount = 100000;
        int years = 25;
        // calculations
        double yearlyIncreaseRatio = getYearlyIncreaseRatio([redacted]);
        double montlyPaymentAmount = getMontlyPaymentAmount([redacted]);
        // display information
        [redacted]
    }
}
```

Output Part 1:



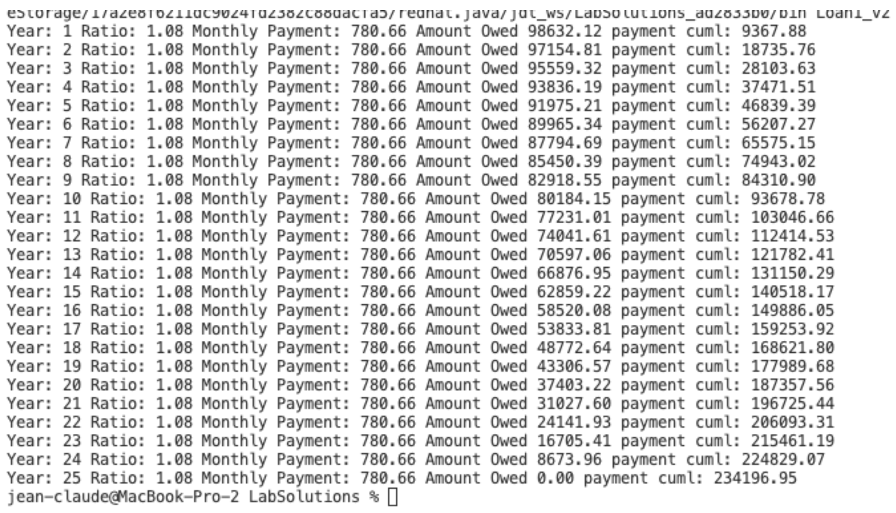
```
Ratio: 1.08 Monthly Payment: 780.66

jean-claude@MacBook-Pro-2 LabSolutions %
```

In Part 2, you are going to use the methods above to continue coding the solution. The goal is to create some code to compute the payments made and the amount owed at a specific year. Then you can port this code into their own methods and use them to display the borrowing details using *printf* method.

Note: The *getLoanCostNoPayment* method is also needed in Part 1, and so is the *yearlyIncreaseRatio* method.

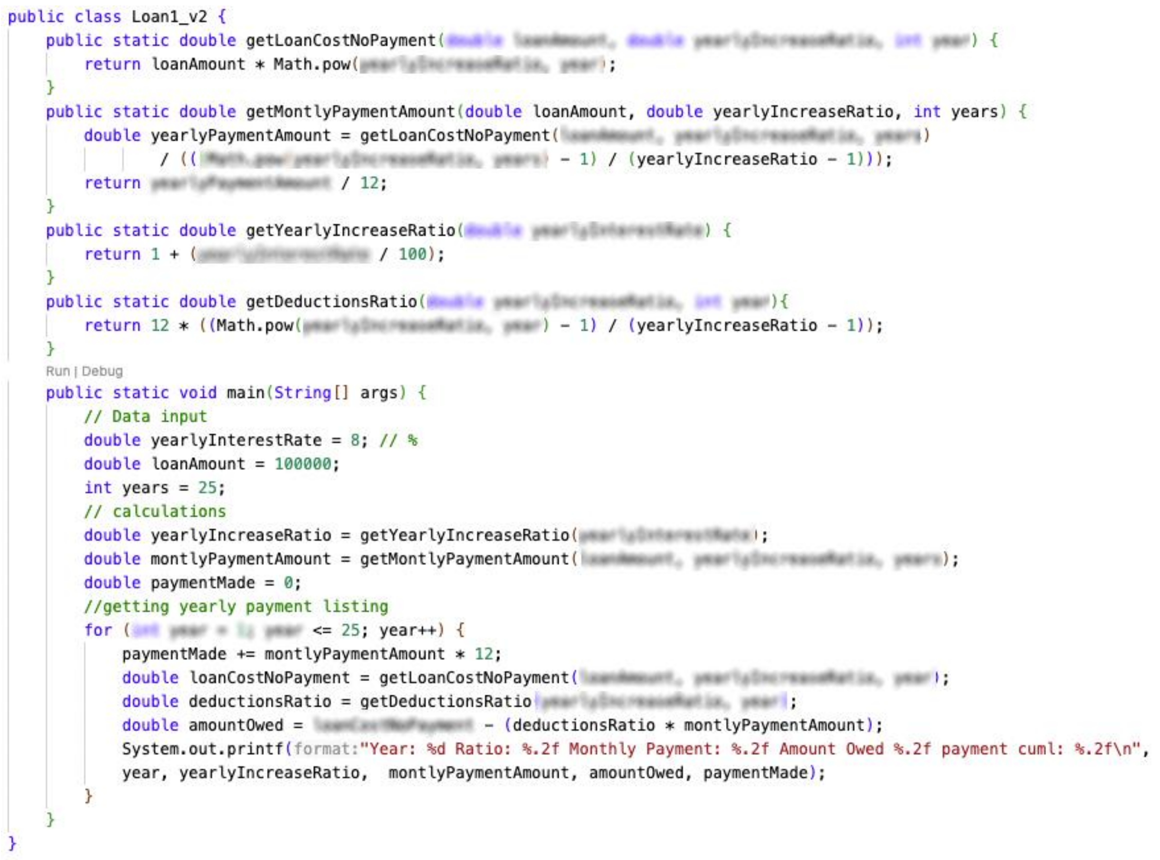
You should use a for loop for the years as shown below.

Output Part 2:



```
eStorage/17a2e8f6211dc9024fd2382c88dacfa5/redhat.java/jdt_ws/LabSolutions_ad2833b0/bin Loan1_v2
Year: 1 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 98632.12 payment cuml: 9367.88
Year: 2 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 97154.81 payment cuml: 18735.76
Year: 3 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 95559.32 payment cuml: 28103.63
Year: 4 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 93836.19 payment cuml: 37471.51
Year: 5 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 91975.21 payment cuml: 46839.39
Year: 6 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 89965.34 payment cuml: 56207.27
Year: 7 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 87794.69 payment cuml: 65575.15
Year: 8 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 85450.39 payment cuml: 74943.02
Year: 9 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 82918.55 payment cuml: 84310.90
Year: 10 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 80184.15 payment cuml: 93678.78
Year: 11 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 77231.01 payment cuml: 103046.66
Year: 12 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 74041.61 payment cuml: 112414.53
Year: 13 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 70597.06 payment cuml: 121782.41
Year: 14 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 66876.95 payment cuml: 131150.29
Year: 15 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 62859.22 payment cuml: 140518.17
Year: 16 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 58520.08 payment cuml: 149886.05
Year: 17 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 53833.81 payment cuml: 159253.92
Year: 18 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 48772.64 payment cuml: 168621.80
Year: 19 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 43306.57 payment cuml: 177989.68
Year: 20 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 37403.22 payment cuml: 187357.56
Year: 21 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 31027.60 payment cuml: 196725.44
Year: 22 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 24141.93 payment cuml: 206093.31
Year: 23 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 16705.41 payment cuml: 215461.19
Year: 24 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 8673.96 payment cuml: 224829.07
Year: 25 Ratio: 1.08 Monthly Payment: 780.66 Amount Owed 0.00 payment cuml: 234196.95
jean-claude@MacBook-Pro-2 LabSolutions % 
```

Code structure for Part 2:



```java
public class Loan1_v2 {
    public static double getLoanCostNoPayment([redacted]) {
        return loanAmount * Math.pow([redacted]);
    }
    public static double getMontlyPaymentAmount(double loanAmount, double yearlyIncreaseRatio, int years) {
        double yearlyPaymentAmount = getLoanCostNoPayment([redacted])
                / (([redacted] - 1) / (yearlyIncreaseRatio - 1)));
        return [redacted] / 12;
    }
    public static double getYearlyIncreaseRatio([redacted]) {
        return 1 + ([redacted] / 100);
    }
    public static double getDeductionsRatio([redacted]){
        return 12 * ((Math.pow([redacted]) - 1) / (yearlyIncreaseRatio - 1));
    }
    Run | Debug
    public static void main(String[] args) {
        // Data input
        double yearlyInterestRate = 8; // %
        double loanAmount = 100000;
        int years = 25;
        // calculations
        double yearlyIncreaseRatio = getYearlyIncreaseRatio([redacted]);
        double montlyPaymentAmount = getMontlyPaymentAmount([redacted]);
        double paymentMade = 0;
        //getting yearly payment listing
        for ([redacted] <= 25; year++) {
            paymentMade += montlyPaymentAmount * 12;
            double loanCostNoPayment = getLoanCostNoPayment([redacted]);
            double deductionsRatio = getDeductionsRatio[redacted];
            double amountOwed = [redacted] - (deductionsRatio * montlyPaymentAmount);
            System.out.printf(format:"Year: %d Ratio: %.2f Monthly Payment: %.2f Amount Owed %.2f payment cuml: %.2f\n",
            year, yearlyIncreaseRatio,  montlyPaymentAmount, amountOwed, paymentMade);
        }
    }
}
```

Appendix

With compounded interest we deal with Geometric series as the as what is added to is based on a ratio. In this exercise this is the case.

For instance, for the geometric series

1 + 3 + 9 + 27

The geometric ratio **r** is 3, as the next digit is the previous one multiplied by 3. We have 4 numbers thus **n** = 4. The initial value **a** is 1. So, we can use the formula below to compute the sum. This is a very simple example but imagine that have hundreds of values!

$$sum = a * \left(\frac{r^n - 1}{r - 1}\right)$$

so, here we would have

$$sum = 1 * \left(\frac{3^4 - 1}{3 - 1}\right) = 40$$

The same applies to our example. We want to sum up values that are multiplied by a ratio. So, in the formula below, the geometric ratio is (1 + monthly interest rate). This is what we call rateIncrease here.

$$New\ Capital\ (Year) = principle * rateIncreaseRatio * \left(\frac{(rateIncreaseRatio^{year} - 1)}{(rateIncreaseRatio - 1)}\right)$$

a	is “principle * rateIncreaseRatio”.
r	is “rateIncrease” (geometric ratio)
n	is year

So, if the rate at which your capital grows by is 1.01 i.e., 1% per month which correspond to a given yearly rate for example of 12% (0.12), which per month is 0.12 /12 months = 0.01, and your initial capital (called principle) is 100, **a** will be 100 * (1+0.01) which is 100 * (1.01). If you want to know how much your principle is after 10 years (if you have not taken money out ☺):

$$Capital\ (after\ 10\ years) = 100 * 1.01 * \left(\frac{(1.01^{10} - 1)}{(1.01 - 1)}\right)$$

$$Capital\ (after\ 10\ years) = 101 * \left(\frac{(1.1046 - 1)}{0.01}\right)$$

$$Capital\ (after\ 10\ years) = 101 * (10.462) = 1056.68$$

After 10 years, your initial capital of £100 will have grown to £1056.68.