Lab Exercises No.2
Accelerated Introduction to Computer Science Fall 2023
Course CS 201 02

Name: Harlee Ramos **Due Date:** 09/08/2023

A - Java Basics - First Program, Compile Errors (1 point each)
Please complete these exercises in a Java development environment.

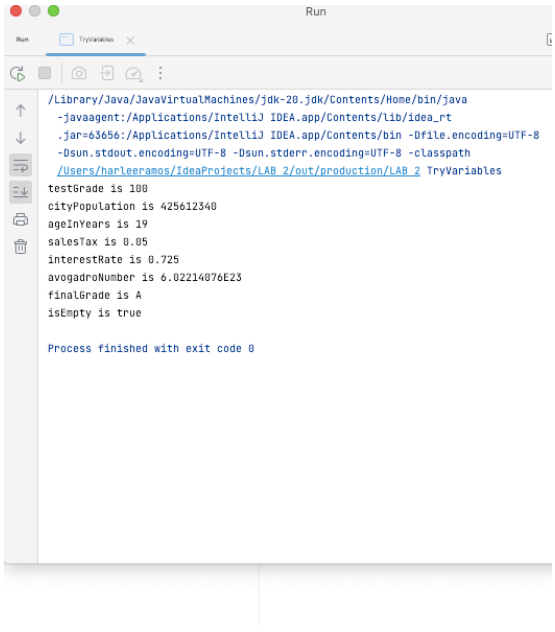
Question 1 - Learn to recognize syntax error messages.
The starter code shown below has two syntax errors that you will find and correct. Try to execute the test case on the code and note the compiler syntax errors. Note: Sometimes a single syntax error can generate more than one compiler error message. Correct the syntax errors and execute again until there are no errors and the expected output is correct.

See TryVariables.java

EXPECTED OUTPUT:

```
testGrade is 100
cityPopulation is 425612340
ageInYears is 19
salesTax is 0.05
interestRate is 0.725
avogadroNumber is 6.02214076E23
finalGrade is A
isEmpty is true
```

Figure 1. Screenshot of corrected TryVariables.java



PROBLEM 1

The bank wants to be sure you can afford to pay them back before they give you a mortgage. One way they consider your ability to repay is by making sure your total debt doesn't exceed a certain percentage of your income, usually 36-42%. This percentage is called the debt ratio. Given your income (a real number) and other monthly debt (a real number), you can use the following formulas to determine the lower and upper limits on your monthly mortgage payments.

Lower limit = (36% of your income) minus your other monthly debt
Upper limit = (42% of your income) minus your other monthly debt

B - Java Basics - Basic Design, Test, Code
For each of the following problems complete an Input-Process-Output Design, Test Table, Java Program.

Question 1a - Input-Process-Output Design - debt ratio (3 points)
Answer the following questions for the below problem.

1. What are the inputs?

For this problem, the inputs are the following:
Monthly Income will be declared as `income`
Other Monthly Debt Payment will be declared as `otherMonthlyDebt`

- **What format are they in? (e.g., integer, floating-point, character, or string).**

Both inputs will be in the format "floating-point double."

- **Any valid or invalid values? (e.g. positive, negative, valid range, certain values).**

Valid values: numbers greater than 0.
Invalid values: numbers less or equal to 0, string values (ex. $1), and character values.

- **How do you get them? (e.g., prompt user or read from file).**

Prompt user

2. How do you get from inputs to the outputs you want (process)?

To obtain input, it is required to instruct the user on how to utilize the program. The program should start with the main instructions; these instructions will be printed on the screen and will give a brief idea of the purpose of the program. The instructions will be:

The initial greeting of the program is:

```
"BANK ABC MORTGAGE CALCULATOR"
"WELCOME"
```

The instruction for the user is:

```
"This application will determine the recommended limits for your new mortgage payments."
```

Then, once the scanner signature and its invocation in the code are complete, the user can be prompted for the desired input. If the scanner class library is not used, it would be necessary to assign values to the input variables in order to provide an output. The program will include the following instructions:

```
"Insert the income amount: " User type text on keyboard.
"Insert the total of other monthly debts amount: " User type text on keyboard.
```

In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output, some of the variables that will contain a set double value are:

```
ratioDebt1= 0.36
ratioDebt2= 0.42
```

After completing those steps, we can start the calculations. By translating our mathematical model for the payments limits in into Java language, the code will be able to execute the desired operation. In this phase, it is important to check with a test chart if the model works or even the logic of the modeling (please refer to table 1). The operations will be:

```
lowLimit = ratioDebt1* income  - otherMonthlyDebt
highLimit = ratioDebt2* income  - otherMonthlyDebt
```

Finally, we will be able to print on the screen the output result.

```
"The lower limit is: $" + lowLimit
"The upper limit is: $" + highLimit
```

a. What are the calculation steps?

We must calculate the lower and the upper limit for determinate the mortgage payments, by the following formulas:

Lower limit = (36% of your income) minus your other monthly debt

```
lowLimit = ratioDebt1* income  - otherMonthlyDebt
```

Upper limit = (42% of your income) minus your other monthly debt

```
highLimit = ratioDebt2* income  - otherMonthlyDebt
```

b. To follow these steps, what else do you need?

1. Decide on the class libraries that will be needed to run the code.
2. Declare the variables.
3. Print the instructions.
4. Build the right mathematical operation to get the results.
5. Print the results.
6. Debug, run, check and fix any possible error in the code.

c. Other variables, constants, conversion? (besides input and output variables)

Variables with constant value:
The debt ratio for the upper limit is equal to 0.36, declared as `ratioDebt1`
The debt ratio for the lower limit is equal to 0.42, declared as `ratioDebt2`

d. Libraries (e.g., for calculations).

The class library for Scanner: `import java.util.Scanner;`

3. What are the outputs?

Lower limit of the monthly mortgage payments
Upper limit of the monthly mortgage payments

- **What format are they in?**

Format floating-point: Double.

- **How do you output them? (e.g., on screen or to a file)**

The output of the value will be on screen.

Question 1b - Test Table - debt ratio (3 points)
Create a test plan with sample data for the below problem. Make a table like this with enough rows. Recall that types of test cases to consider:

- for each different possible output.
- for either side of a boundary, and on the boundary, implied in the problem.
- for common known answers (like 32 F = 0 C).
- for low, average and high input values.

At this time, we are not writing test cases for invalid input values. We are just identifying those in the Input-Process-Output design.

Table 1. Test table for problem 1

Test Case Reason	Sample Data		Expected Result (manually calculate)	
otherMonthlyDebt <= 0 && income > 0	Other Monthly Debts	0	Lower Limit	1800
	Income	5000	Upper Limit	2100
income > otherMonthlyDebt	Other Monthly Debts	1000	Lower Limit	800
	Income	5000	Upper Limit	1100
income = 360	Other Monthly Debts	0	Lower Limit	129.6
	Income	360	Upper Limit	151.2
income = 420	Other Monthly Debts	0	Lower Limit	151.2
	Income	420	Upper Limit	176.4
otherMonthlyDebt = 0	Other Monthly Debts	0	Lower Limit	360
	Income	1000	Upper Limit	420
income = 0	Other Monthly Debts	10000	Lower Limit	-10000
	Income	0	Upper Limit	-10000
income = otherMonthlyDebt	Other Monthly Debts	6000	Lower Limit	-3840
	Income	6000	Upper Limit	-3480
otherMonthlyDebt > income	Other Monthly Debts	5000	Lower Limit	-4640
	Income	1000	Upper Limit	-4580
otherMonthlyDebt = 360	Other Monthly Debts	360	Lower Limit	-360
	Income	0	Upper Limit	-360
otherMonthlyDebt = 420	Other Monthly Debts	420	Lower Limit	-60
	Income	1000	Upper Limit	0

Conclusion

The test table method analysis was used to determine the possible values that the user can enter into the program, then the values must satisfy the following conditions:

The income must be greater than 0, greater than the other monthly debts, and the other monthly debts can be greater than or equal to 0.

$$otherMonthlyDebt < income \;\&\&\; income > 0 \;\&\&\; otherMontlhyDebt \geq 0$$

Question 1c - Write Code - debt ratio (3 points)
Implement your Input-Process-Output Design in Java. Make sure to utilize these programming concepts correctly when appropriate:

- comments
- descriptive identifier names
- correct data types
- unit conversion
- String literals for output using `System.out.println()`
- constants
- order of operations
- mixed type arithmetic (implicit type casting)
- % operator (mod or remainder)
- explicit type casting

Since we have not yet covered user input, you are provided with starter code which will read user input from in the test cases provided. The starter code also provides the output statements for the test cases. Make sure to test your code on all the test cases provided.

EXPECTED OUTPUT (for various inputs):

```
Test Case 1 - avg income, avg debt    4000   1000
Lower Limit: $440.0
High Limit: $680.0

Test Case 2 - high income, low debt   6000   500
Lower Limit: $1660.0
High Limit: $2020.0

Test Case 3 - zero debt   3000   0
Lower Limit: $1080.0
High Limit: $1260.0

Test Case 4 - low income, high debt   2000   700
Lower Limit: $20.0
High Limit: $140.0
```

Figure 2. Problem 1: Test Case 1

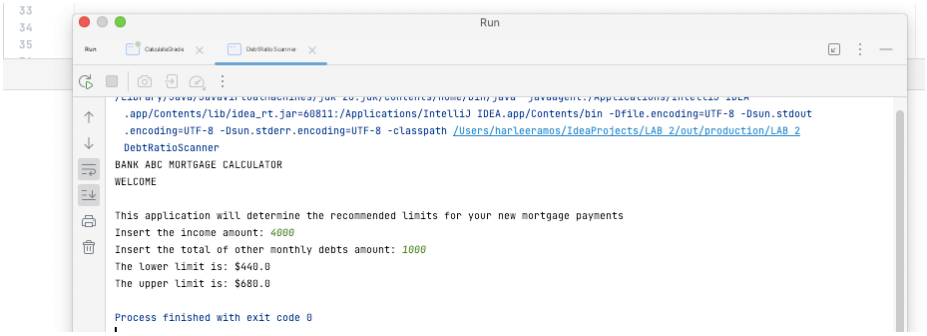


Figure 3. Problem 1: Test Case 2

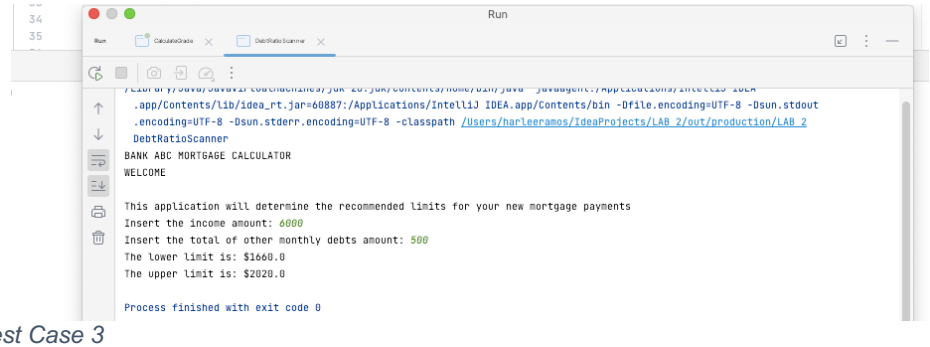


Figure 4. Problem 1: Test Case 3

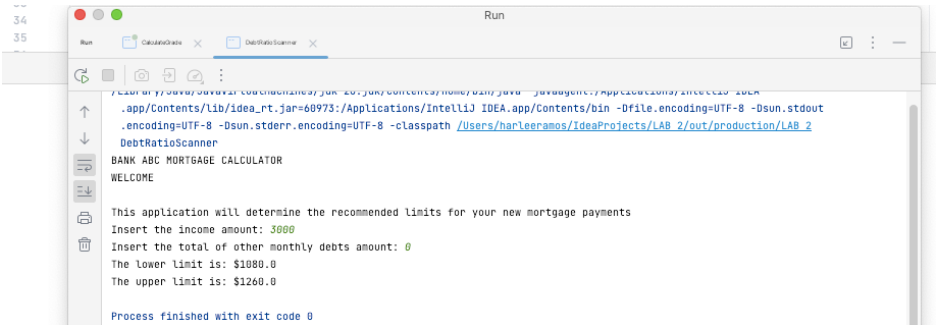
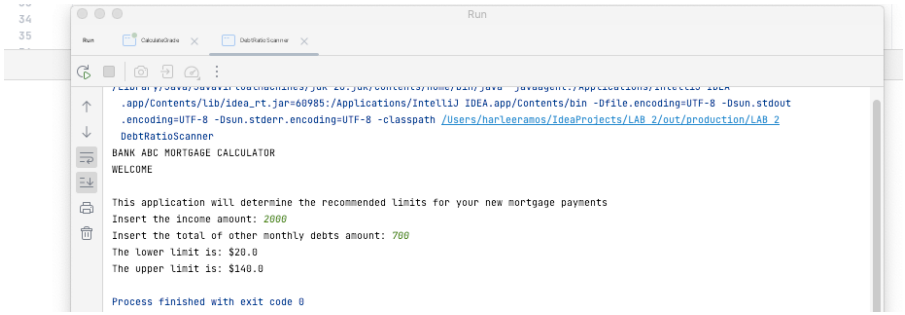


Figure 5. Problem 1: Test Case 4


Run
.app/Contents/lib/idea_rt.jar=60985:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout
.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /Users/harleeramos/IdeaProjects/LAB 2/out/production/LAB 2
DebtRatioScanner
BANK ABC MORTGAGE CALCULATOR
WELCOME
This application will determine the recommended limits for your new mortgage payments
Insert the income amount: 2000
Insert the total of other monthly debts amount: 700
The lower limit is: $20.0
The upper limit is: $140.0
Process finished with exit code 0

PROBLEM 2

Some private water wells produce only 1/3 of a gallon of water per minute. Each member of a family will use around 60 gallons of water per day. One way to avoid running out of water with these low-yield wells is to use a separate, above-ground holding tank. However, there is also a "natural" water holding tank in the casing (i.e. the cylindrical hole) of the well itself. The deeper the well, the more water that will be stored in the casing that can be pumped out for household use. But water will not fill the entire depth of the well, in practice the static water level will generally be 50 feet below the ground surface.

We want to calculate the "tankSize" which is the minimum size of the separate, above-ground holding tank so there will be enough water in the well and the above-ground holding tank. at the start of the day for one day's use. The three inputs are:

- The real number radius of the well casing in inches (usually 2-6 inches).
- The real number total depth of the well in feet (usually 50-250 feet).
- The number of family members.

3a. Input-Process-Output Design - tank size

Answer the following questions for the below problem. Type your answers in the text box:

1. What are the inputs?

For this problem, the inputs are the following:

- The real number radius of the well casing in inches, it will be declared as `radius`
- The real number total depth of the well in feet, it will be declared as `depth`
- The number of family members, it will be declared as `familySize`

- **What format are they in? (e.g., integer, floating-point, character, or string).**

For the input `radius` the format is *floating-point: double.*
For the input depth the format is *floating-point: double.*
For the input `familySize` "the format is *integer.*

- **Any valid or invalid values? (e.g. positive, negative, valid range, certain values).**

Valid values: Numbers greater than 0.
Invalid values: Numbers less or equal than 0, string values and character values.

- **How do you get them? (e.g., prompt user or read from file).**

Prompt user

2. How do you get from inputs to the outputs you want (process)?

To obtain input, it is required to instruct the user on how to utilize the program. The program should start with the main instructions; these instructions will be printed on the screen and will give a brief idea of the purpose of the program. The initial greeting of the program:

```
"WELCOME!"
```

The instruction for the user:

```
"This program will help you determine the tank size that you need."

"In order to determine the size, you should know the radius of the well in inches, the
    depth of the well in feet, and the number of family members."
```

Then, once the scanner signature and its invocation in the code are complete, the user can be prompted for the desired input. If the scanner class library is not used, it would be necessary to assign values to the input variables in order to provide an output. The program will include the following instructions:

```
"Input the radius: " User type text on keyboard.

"Input the depth: " User type text on keyboard.
"Input the quantity of family members: " User type text on keyboard.
```

In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output.

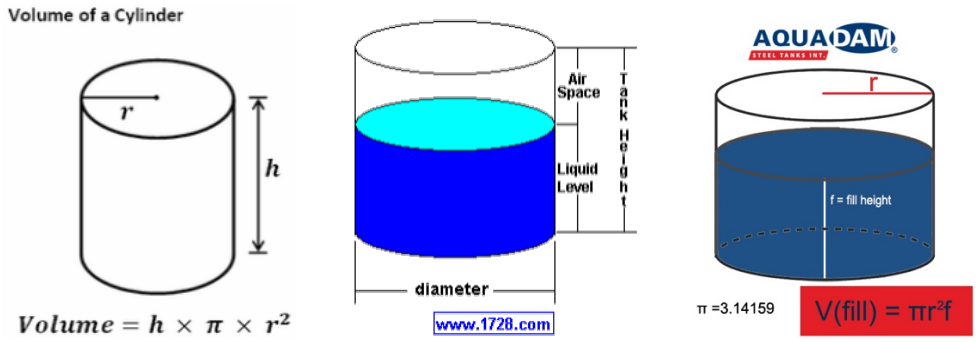
After completing those steps, we can start the calculations. By translating our mathematical model into Java, the code will be able to execute the desired operation. In the next question, the steps will be explained. In this phase, it is important to check with a test chart if the model works or even the logic of the modeling (please refer to Table 2).

Finally, we will be able to print on the screen the output result.

```
"Tank Size Needed = " + gallonsNeeded  value printed on screen +"gal. "
```

- **What are the calculation steps?**

To get the value of the output, it would be necessary to calculate the following: calculation of the volume of the well. Note that since the well has a cylindrical shape, the mathematical model of the volume in this shape was used.



Source: Google Images

The code will be written in this way:

```
wellVolume=  (shotPIvalue)*(depth-waterLevel)*(Math.pow((radius/inch2foot),2))
```

Then the value of the well volume needs to be converted to gallons; for this reason, in the following operation, the well volume is multiplied by the conversion of feet to gallons.

```
wellGallon= wellVolume*cubFoots2gal
```

Next, the gallons per family member are determined by the product of the given constant gallon per person and the family size.

```
famGallon = galXperson*familySize
```

Finally, the total gallon needed is the sum of the well gallons plus the family gallons.

```
gallonsNeeded = famGallon - wellGallon
```

- **To follow these steps, what else do you need?**

1. Decide on the class libraries that will be needed to run the code.
2. Declare the variables.
3. Print the instructions.
4. Build the right mathematical operation to get the results.
5. Print the results.
6. Debug, run, check and fix any possible error in the code.

- **Other variables, constants, conversion? (besides input and output variables)**

For the operations of the process, it will be required to set some constants for conversions of units.
Constants:

- `inch2foot`. Variable with set value: Conversion of inch to foot, 1 inch is equal to 12 foot.
- `cubFoots2gal`. Variable with set value: Conversion of cubic foot to gallons, 1 cubic foot is equal to 7.48 gallons.

- `waterLevel`. Variable with set value: Water level equal to 50 gallons.
- `galXperson`. Variable with set value: Gallons per person equal to 60 gallons.
- `shotPIvalue`. Variable with set value for express the π value: 3.14159.
- Mathematical method Math.pow();

- **Libraries (e.g., for calculations).**

The class library for Scanner: `import java.util.Scanner;`

3. What are the outputs?

The tank size needed

- **What format are they in?**

Floating point: double

- **How do you output them? (e.g., on screen or to a file)**

The output of the value will be on screen.

- **What does it mean if the calculated above-ground holding tank size is negative?**

If the calculated value is negative, it means that the gallons in the well are higher than the gallons per family. In this case, it would be necessary to set conditions in the code that could prevent the error.

3b. Test Table - tank size

Create a test plan with sample data for the below problem. Make a table like this with enough rows. Recall that types of test cases to consider:

- for each different possible output
- for either side of a boundary, and on the boundary, implied in the problem
- for common known answers (like 32 F = 0 C)
- for low, average and high input values

At this time, we are not writing test cases for invalid input values. We are just identifying those in the Input-Process- Output design.

Table 2. Test table for problem 2.

Test Case Reason	Sample Data		Expected Result (manually calculate)	
radious < depth, familySize	radious	3.5	gallonsNeeded	200.05
	depth	100		
	family size	5		
depth < radious, familySize	radious	100	gallonsNeeded	54356.44
	depth	20		
	family size	90		
familySize > radious, depth	radious	10	gallonsNeeded	23184.06
	depth	100		
	family size	400		
radious =< 0 && radious < depth, familySize	radious	0	gallonsNeeded	300.00
	depth	100		
	family size	5		
depth =< 0 && depth < depth, familySize	radious	100	gallonsNeeded	86994.07
	depth	0		
	family size	90		
familySize =< radious && depth	radious	10	gallonsNeeded	-8975.35
	depth	600		
	family size	0		
radious > depth && familySize	radious	100	gallonsNeeded	-65155.26
	depth	90		
	family size	2		
depth > radious && familySize	radious	100	gallonsNeeded	-897174.81
	depth	600		
	family size	6		
familySize < radious && depth	radious	10	gallonsNeeded	-8915.35
	depth	600		
	family size	1		

familySize = radious =depth=0	radious	0	gallonsNeeded	0.00
	depth	0		
	family size	0		
familySize = radious =depth	radious	0	gallonsNeeded	3600.00
	depth	0		
	family size	60		
familySize = radious =0 && depht > 0	radious	0	gallonsNeeded	0.00
	depth	100		
	family size	0		
radious =depth =0 && familySize > 0	radious	0	gallonsNeeded	60.00
	depth	0		
	family size	1		
familySize = radious =0 && depht < 0	radious	0	gallonsNeeded	0.00
	depth	1		
	family size	0		
radious =depth =0 && familySize < 0	radious	1	gallonsNeeded	8.16
	depth	0		
	family size	0		

Conclusion

The test table method analysis was used to determine the possible values that the user can enter into the program, then the values must satisfy the following conditions: The radius, the depth and the familySize must be higher than cero, the depth cannot be higher than the other two inputs, likewise the familySize cannot be higher than the two inputs.

3c. Write Code - tank size

Implement your Input-Process-Output Design in Java. Make sure to utilize these programming concepts correctly when appropriate:

- comments
- descriptive identifier names
- correct data types
- unit conversion
- String literals for output using System.out.println()
- constants
- order of operations
- mixed type arithmetic (implicit type casting)
- % operator (mod or remainder)
- • explicit type casting

Since we have not yet covered user input, you are provided with starter code which will read user input from in the test cases provided. The starter code also provides the output statements for the test cases.
Make sure to test your code on all the test cases provided.

EXPECTED OUTPUT (for various inputs):

```
Test Case 1 - large tank needed          3.5   125   5
Tank Size Needed = 150.0708897395833 gal.

Test Case 2 - no tank needed (calculated tank size negative)       6   200   2
Tank Size Needed = -761.215995 gal.

Test Case 3 - no tank needed (calculated tank size close to zero)        3.5   140   3
Tank Size Needed = 0.0850676875000147 gal.
```

Figure 6. Problem 2: Test Case 1

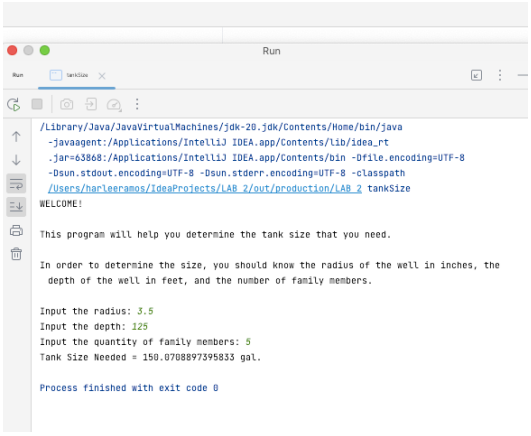


Figure 7. Problem 2: Test Case 2

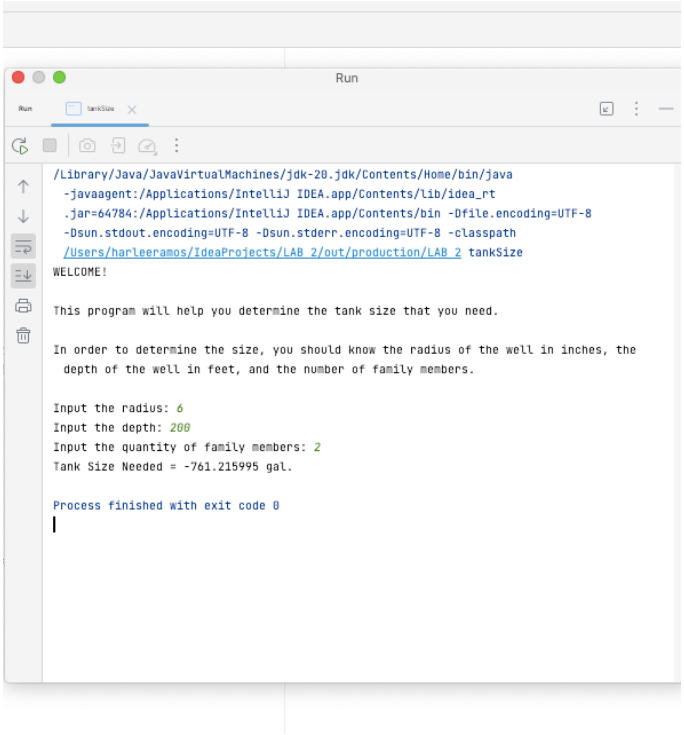
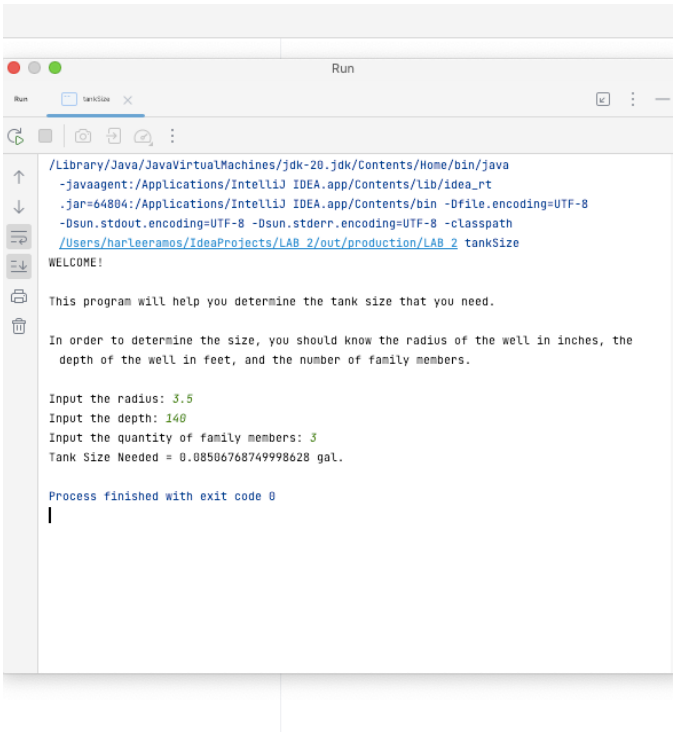


Figure 8. Problem 2: Test Case 3



PROBLEM 3

Most stop watches allow you to display the time elapsed as a number of seconds, or as hours:minutes: seconds. So, 5437 seconds or 1 hour:30 minutes:37 seconds Given an integer number of seconds, calculate the equivalent integer number of hours, integer number of minutes, and integer number of seconds.

Question 2a - Input-Process-Output Design – stopwatch (3 points)

Answer the following questions for the below problem. Type your answers in the text box:

1. What are the inputs?

For this problem the input is

Integer number of second `seconds`

- **What format are they in? (e.g., integer, floating-point, character, or string).**

Integer: Integer

- **Any valid or invalid values? (e.g. positive, negative, valid range, certain values).**

Valid values: Numbers greater and equals than 0.
Invalid values: Numbers less than 0, string values, floating- point values and character values.

- **How do you get them? (e.g., prompt user or read from file).**

Prompt user

1. How do you get from inputs to the outputs you want (process)?

To obtain input, it is required to instruct the user on how to utilize the program. The program should start with the main instructions; these instructions will be printed on the screen and will give a brief idea of the purpose of the program.

The initial greeting of the program:

```
"WELCOME"
"This program will convert an integer format time to hh:mm:ss"
```

Then, once the scanner signature and invocation its invocation in the code are complete, the user can be prompted for the desired input. If the scanner class library is not used, it would be necessary to assign values to the input variables in order to provide an output. In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output.

The instruction for the user:

```
"Insert the integer time: " User type text on keyboard.
```

After completing those steps, we can start the calculations. By translating our mathematical model into Java, the code will be able to execute the desired operation. In this phase, in this phase, it is important to check with a test chart if the model works or even the logic of the modeling (please refer to table 3). The operations will be:

```
hours =seconds/(min2second*hour2min)
    leftOver = seconds%min2second
     minutes=leftOver/min2second

    leftOver = seconds%min2second
```

Finally, we will be able to print on the screen the output result.

```
seconds+" seconds equals " + hours + ":" + minutes + ":" + leftOver
```

- **What are the calculation steps?**

In order to calculate the conversion of an integer number of seconds into hours, minutes and seconds, it is required to perform the following operations:

To convert the seconds to hours, with this operation is the same as expressing that 1 hour is equal to 3600 seconds (60x60).

```
hours =seconds/(min2second*hour2min)
```

The modulus can be used to recover the remainder of an integer division: seconds divided by 60 may not be divisible, therefore the% helps to save the remaining of the las operation.

```
The left-over of the hours is expressed as:
leftOver = seconds%min2second
```

After having the remaining value in seconds, it is possible to convert the value in minutes, and then the leftover value is divided by 60 seconds.

```
minutes=leftOver/min2second
```

Then again, the last operation has remains, the left-over of the minutes is expressed as:

```
leftOver = seconds%min2second
```

- **To follow these steps, what else do you need?**

1. Decide on the class libraries that will be needed to run the code.
2. Declare the variables.
3. Print the instructions.
4. Build the right mathematical operation to get the results.
5. Print the results.
6. Debug, run, check and fix any possible error in the code.

- **Other variables, constants, conversion? (besides input and output variables)**

Conversion of 1 minutes equal to 60 seconds `min2second =60`
Conversion of 1 hour equal to 60 minutes `hour2min =60`

- **Libraries (e.g., for calculations).**

The class library for Scanner: `import java.util.Scanner;`

4. What are the outputs?

Integer number in hours, `hours`
Integer number in minutes, `minutes`
Integer number in seconds, `leftOver`

- **What format are they in?**

Format Integer: Integer

- **How do you output them? (e.g., on screen or to a file)**

The output of the value will be on screen.

Question 2b - Test Table – stopwatch (3 points)
Create a test plan with sample data for the below problem. Make a table like this with enough rows. Recall that types of test cases to consider:

- for each different possible output
- for either side of a boundary, and on the boundary, implied in the problem
- for common known answers (like 32 F = 0 C)
- for low, average and high input values
  At this time, we are not writing test cases for invalid input values. We are just identifying those in the Input-Process-Output design.

Table 3. Test table for problem 3

Test Case Reason	Sample Data		Expected Result (manually calculate)	
seconds = 0	seconds	0	hours	0
			minutes	0
			seconds	0
seconds > 0	seconds	8990	hours	2
			minutes	29
			seconds	50

Question 2c - Write Code – stopwatch (3 points)
Implement your Input-Process-Output Design in Java. Make sure to utilize these programming concepts correctly when appropriate:

- comments
- descriptive identifier names
- correct data types
- unit conversion
- String literals for output using `System.out.println()`
- constants
- order of operations
- mixed type arithmetic (implicit type casting)
- % operator (mod or remainder)
- explicit type casting

Since we have not yet covered user input, you are provided with starter code which will read user input from in the test cases provided. The starter code also provides the output statements for the test cases. Make sure to test your code on all the test cases provided. Most stop watches allow you to display the time elapsed as a number of seconds, or as hours:minutes: seconds. So 5437 seconds or 1 hour:30 minutes:37 seconds Given an integer number of seconds, calculate the equivalent integer number of hours, integer number of minutes, and integer number of seconds.
See Stopwatch.java

EXPECTED OUTPUT (for various inputs):

```
324
324 seconds equals 0:5:24

4127
4127 seconds equals 1:8:47

3635
3635 seconds equals 1:0:35

475
475 seconds equals 0:7:55

7260
7260 seconds equals 2:1:0

12345
12345 seconds equals 3:25:45
```

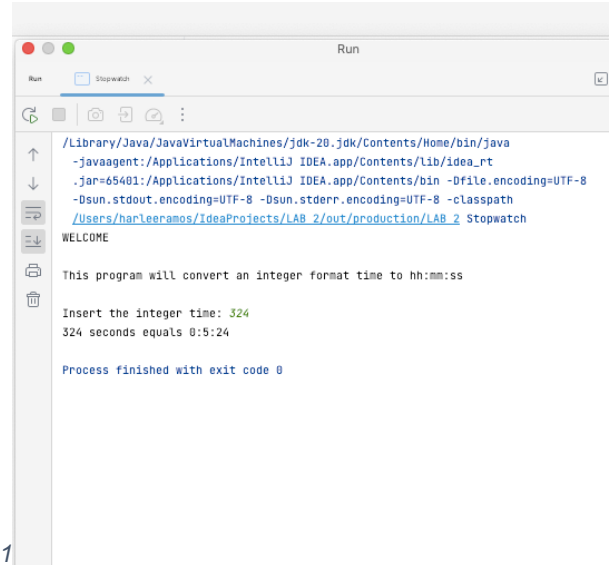


Figure 9. Problem 3: Test Case 1

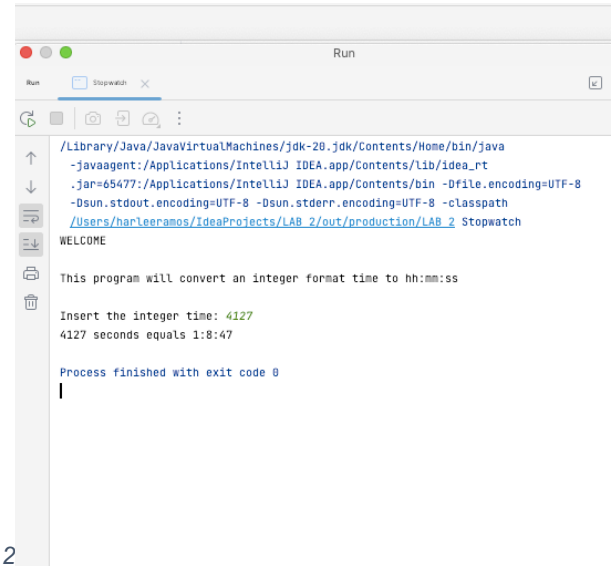


Figure 10. Problem 3: Test Case 2

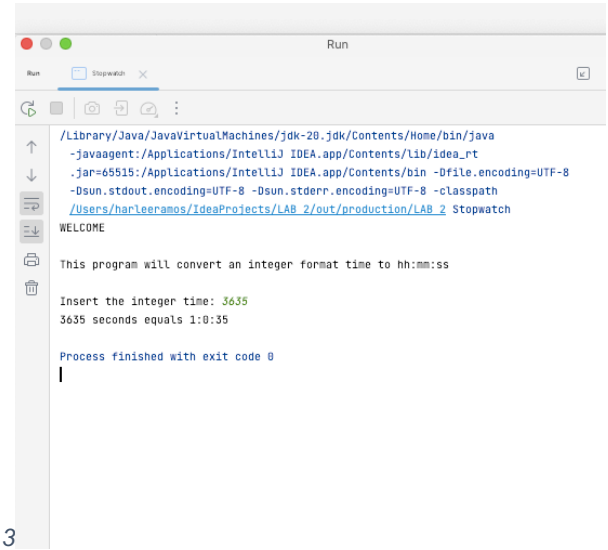


Figure 11. Problem 3: Test Case 3

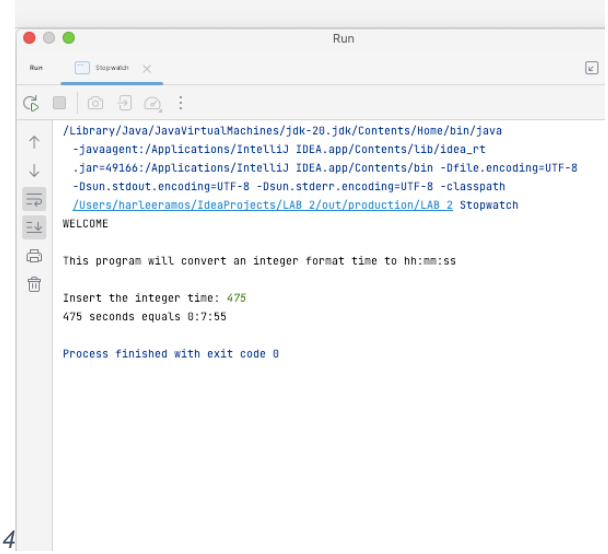


Figure 12. Problem 3: Test Case 4

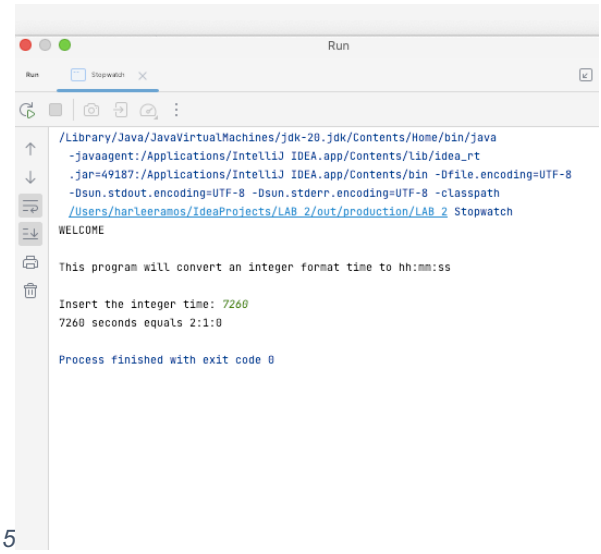


Figure 13. Problem 3: Test Case 5

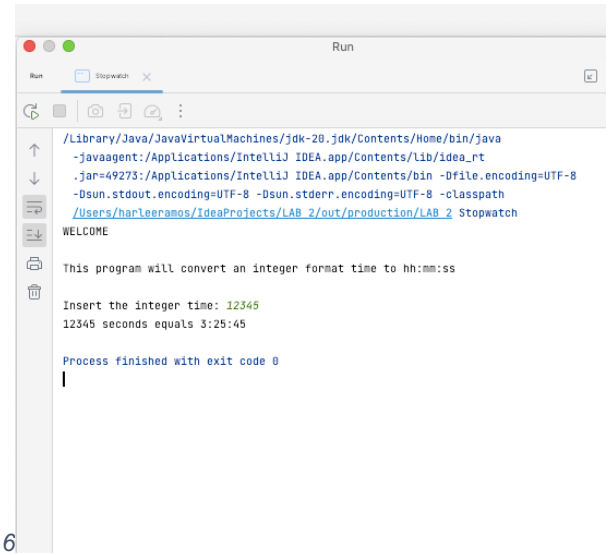


Figure 14. Problem 3: Test Case 6

PROBLEM 4

The CSxxx course has the following weighting of the course assignments:
Quiz #1 10%
Quiz #2 15%
Midterm 25%
Final 30%
Labs 20%
TOTAL 100%

The instructor would like a program to calculate a student's weighted average. The assignment grades are entered as integers (in the above order), and the calculated student's weighted average should be decimal.
Question 4a - Input-Process-Output Design - calculate grade
Answer the following questions for the below problem. Type your answers in the text box:

1. What are the inputs?

Grade of the quiz 1
Grade of the quiz 2
Grade of the midterm
Grade of the final exam
Grade of the laboratory

- **What format are they in? (e.g., integer, floating-point, character, or string).**

Floating point: double

- **Any valid or invalid values? (e.g. positive, negative, valid range, certain values).**

Valid values: Numbers greater or equal than 0.
Invalid values: Numbers less than 0, string values and character values.

- **How do you get them? (e.g., prompt user or read from file).**

Prompt user

2. How do you get from inputs to the outputs you want (process)?

To obtain input, it is required to instruct the user on how to utilize the program. The program should start with the main instructions; these instructions will be printed on the screen and will give a brief idea of the purpose of the program.
The initial greeting of the program:

```
"WELCOME"
"This program calculate the weighted grade of the course CS201,02"
```

The instructions for the user:

```
"Type student name: "
"Type quiz N°1 grade: "
"Type quiz N°2 grade: "
"Type midterm grade: "
"Type final exam grade: "
"Type laboratory grade: "
```

Then, once the scanner signature and invocation its invocation in the code are complete, the user can be prompted for the desired input. If the scanner class library is not used, it would be necessary to assign values to the input variables in order to provide an output. In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output.

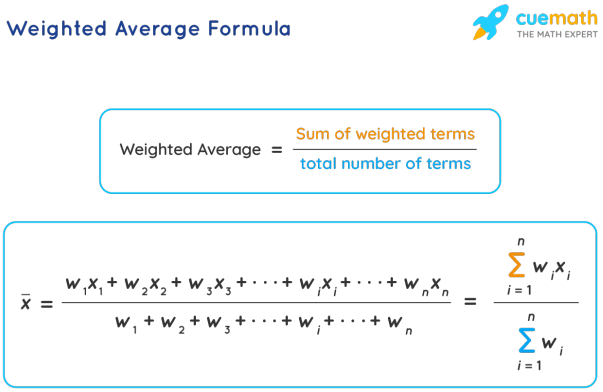
```
weightedGrade=(quiz1*quiz1Weight)+ (quiz2*quiz2Weight)+
(midterm*midtermWei)+(finalExam*finalExamWei)+(labGrades*labGradesWei)
```

After completing those steps, we can start the calculations. By translating our mathematical model into Java, the code will be able to execute the desired operation. In this phase, it is important to check on paper if the model works or even the logic of the modeling.

Finally, we will be able to print on the screen the output result.

3. What are the calculation steps?

In order to calculate the weighted average, it is required to use the weighted average formula



In terms of Java language, the operation for the weighted average will be represented as:

```
weightedGrade=(quiz1*quiz1Weight)+(quiz2*quiz2Weight)+(midterm*midtermWei)+(finalExam*finalExamWei)+(labGrades*labGradesWei)
```

a. To follow these steps, what else do you need?

1. Decide on the class libraries that will be needed to run the code.
2. Declare the variables.
3. Print the instructions.
4. Build the right mathematical operation to get the results.
5. Print the results.
6. Debug, run, check and fix any possible error in the code.

- **Other variables, constants, conversion? (besides input and output variables)**

The problem suggested to use the following percentages that will represent the weights of each grade, it will be necessary to convert the percentages in numbers, so in the code it will be expressed as %/100%.

Values of the problem		Constants in the code
Quiz #1	10%	quiz1Weight= 0.10
Quiz #2	15%	quiz2Weight= 0.15
Midterm	25%	midtermWei=0.25
Final	30%	finalExamWei =0.30
Labs	20%	labGradesWei=0.20
TOTAL	100%	TOTAL 1

- **Libraries (e.g., for calculations).**

The class library for Scanner: `import java.util.Scanner;`

What are the outputs?

Weighted average

- **What format are they in?**

Format floating-point: Double.

- **How do you output them? (e.g., on screen or to a file)**

The output of the value will be on screen.

Question 4b - Test Table - calculate grade
Create a test plan with sample data for the below problem. Make a table like this with enough rows. Recall that types of test cases to consider:

- for each different possible output
- for either side of a boundary, and on the boundary, implied in the problem
- for common known answers (like 32 F = 0 C)
- for low, average and high input values

At this time we are not writing test cases for invalid input values. We are just identifying those in the Input-Process-Output design.

Table 4. Test table for problem 4

Test Case Reason	Sample Data		Expected Result (manually calculate)	
quiz1>= quiz2, midterm, finalExam, labGrades	quiz1	100	weightedGrade	54.45
	quiz2	89		
	midterm	20		
	finalExam	47		
	labGrades	60		
quiz1<=0 && quiz1< quiz2, midterm, finalExam, labGrades	quiz1	0	weightedGrade	90.00
	quiz2	100		
	midterm	100		
	finalExam	100		
	labGrades	100		
quiz2>= quiz1, midterm, finalExam, labGrades	quiz1	90	weightedGrade	68.80
	quiz2	89		
	midterm	35		
	finalExam	67		
	labGrades	88		
quiz2<=0 && quiz2< quiz1, midterm, finalExam, labGrades	quiz1	100	weightedGrade	85.00
	quiz2	0		
	midterm	100		
	finalExam	100		
	labGrades	100		
midterm>= quiz1, quiz2, finalExam, labGrades	quiz1	90	weightedGrade	68.80
	quiz2	89		
	midterm	35		
	finalExam	67		
	labGrades	88		
midterm<=0 && midterm< quiz1, quiz2, finalExam, labGrades	quiz1	100	weightedGrade	85.00
	quiz2	0		
	midterm	100		
	finalExam	100		
	labGrades	100		
labGrades> = quiz1, quiz 2, midterm, finalExam	quiz1	56	weightedGrade	55.85
	quiz2	90		
	midterm	43		
	finalExam	20		
	labGrades	100		
labGrades<=0 && labGrades< quiz1, quiz 2, midterm, finalExam	quiz1	100	weightedGrade	80.00
	quiz2	100		
	midterm	100		
	finalExam	100		
	labGrades	0		
finalExam>= quiz1, quiz 2, midterm, labGrades	quiz1	73	weightedGrade	60.75
	quiz2	70		
	midterm	29		

	finalExam	99		
	labGrades	30		
finalExam<=0 && finalExam, quiz1, quiz 2, midterm, labGrades	quiz1	100	weightedGrade	70.00
	quiz2	100		
	midterm	100		
	finalExam	0		
	labGrades	100		
(quiz1 = quiz 2 = midterm = labGrades = finalExam)>=0	quiz1	90	weightedGrade	90.00
	quiz2	90		
	midterm	90		
	finalExam	90		
	labGrades	90		

Conclusion

As a conclusion for the data obtained by the chart test, it is possible to determine that the value of all the inputs must be higher or equal to cero; however, it is important to highlight that if the sum of the grades of the midterm and the final exam represents more than 50% of the final grade, then these values are the ones that have the greatest influence on the final output.

Question 4c - Write Code - calculate grade
Implement your Input-Process-Output Design in Java. Make sure to utilize these programming concepts correctly when appropriate:

- comments
- descriptive identifier names
- correct data types
- unit coversion
- String literals for output using `System.out.println()`
- constants
- order of operations
- mixed type arithmetic (implicit type casting)
- % operator (mod or remainder)
- explicit type casting

Since we have not yet covered user input, you are provided with starter code which will read user input from in the test cases provided. The starter code also provides the output statements for the test cases. Make sure to test your code on all the test cases provided.

EXPECTED OUTPUT

Test Case 1 - all inputs identical
80 80 80 80 80
Weighted Average = 80.0

Test Case 2 - all passing grades
100 95 90 85 80
Weighted Average = 88.25

Test Case 3 - one zero grade
0 90 91 92 93 94
Weighted Average = 82.44999999999999

Test Case 4 - two zero grades
70 0 80 90 0
Weighted Average = 54.0

Test Case 5 - three zero grades
0 0 0 100 100
Weighted Average = 50.0

Test Case 6 - all zero grades
0 0 0 0 0
Weighted Average = 0.0

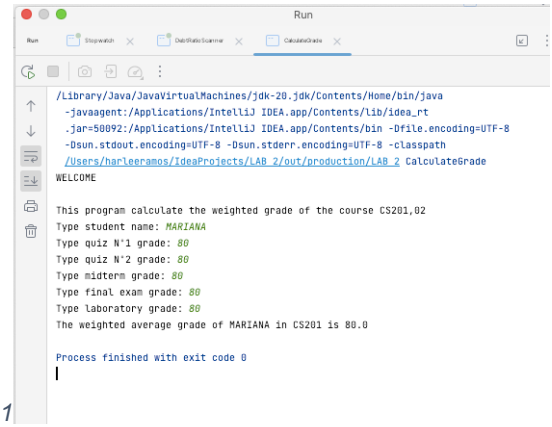


Figure 15. Problem 4: Test Case 1

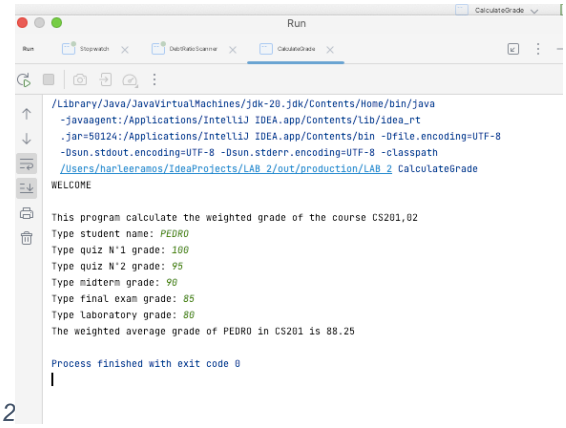


Figure 16. Problem 4: Test Case 2

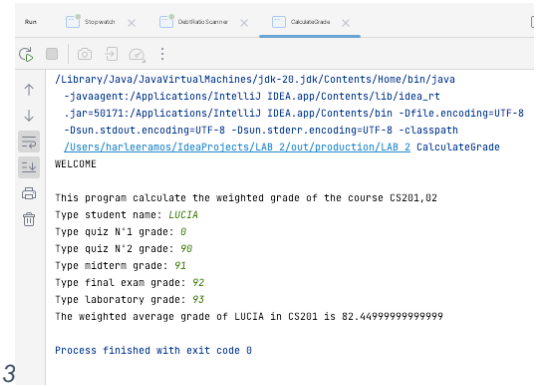


Figure 17. Problem 4: Test Case 3

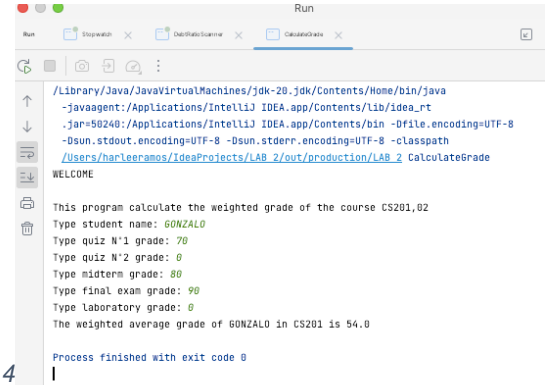


Figure 18. Problem 4: Test Case 4

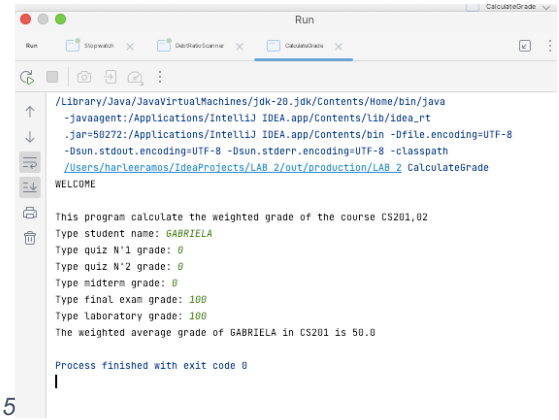


Figure 19. Problem 4: Test Case 5

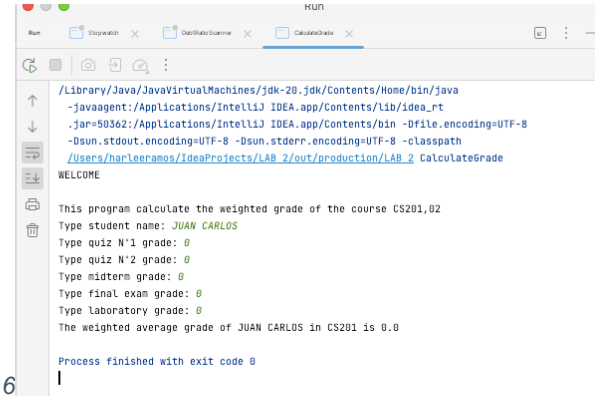


Figure 20. Problem 4: Test Case 6

PROBLEM 5

Professional triathletes often use a heart monitor to check their heart rates during training and competition. The monitor beeps a warning tone if the heart rate is not within an acceptable preset range of values. The weekend athlete can use a pair of simple formulas based solely on age to determine an acceptable heart rate when exercising aerobically. This heart rate is expressed as a range of values called the individual target heart rate zone (THRZ). To calculate your predicted maximal heart rate, subtract your age from 220. Then use the following formulas to determine the limits on your THRZ in beats per minute (bpm).

Lower limit (bpm) = 60% of the difference between 220 and your age
Upper limit (bpm) = 75% of the difference between 220 and your age

Write a program that calculates and outputs a person's target heart rate range depending on their age.

Question 5a - Input-Process-Output Design - target heart rate
Answer the following questions for the below problem. Type your answers in the text box:

2. What are the inputs?

For this problem, the input is:
User's age will be declared as `UserAge`

- **What format are they in? (e.g., integer, floating-point, character, or string).**

The input will be in the format integer.

- **Any valid or invalid values? (e.g. positive, negative, valid range, certain values).**

Valid values: Numbers greater or equal than 0.
Invalid values: Numbers less or equal than 0, string values and character values.

- **How do you get them? (e.g., prompt user or read from file).**

Prompt user

2. How do you get from inputs to the outputs you want (process)?

To obtain input, it is required to instruct the user on how to utilize the program. The program should start with the main instructions; these instructions will be printed on the screen and will give a brief idea of the purpose of the program. The initial greeting of the program is:

```
"WELCOME"
```

The instruction for the user is:

```
"Based on your age, this program will calculate your predicted target heart rate
range. "
```

Then, once the scanner signature and invocation its invocation in the code are complete, the user can be prompted for the desired input. If the scanner class library is not used, it would be necessary to assign values to the input variables in order to provide an output. In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output. The program will include the following instructions:

```
"Type your age: " User type text on keyboard.
```

In addition, it will be necessary to declare variables with set values or constants that will ease the formula for the desired output. After completing those steps, we can start the calculations. By translating our mathematical model into

Java, the code will be able to execute the desired operation. In this phase, it is important to check on paper if the model works or even the logic of the modeling.

```
lowthr = lowLidiff*(220-userAge)

highthr = upperLidiff*(220-userAge)
```

Finally, we will be able to print on the screen the output result.

```
"THRZ: " + lowthr + " - " + highthr + " BPM"
```

a. What are the calculation steps?

We must calculate the lower and upper limits for determining the mortgage payments using the following formulas:
`lowthr = lowLidiff*(220-userAge)`
`highthr = upperLidiff*(220-userAge)`

b. To follow these steps, what else do you need?

1. Decide on the class libraries that will be needed to run the code.
2. Declare the variables.
3. Print the instructions.
4. Build the right mathematical operation to get the results.
5. Print the results.
6. Debug, run, check and fix any possible error in the code.

c. Other variables, constants, conversion? (besides input and output variables)

Variables with constant value:
`lowLidiff=0.6`
`upperLidiff=0.75`

d. Libraries (e.g., for calculations).

The class library for Scanner: `import java.util.Scanner;`

3. What are the outputs?

Lower limit (bpm) = 60% of the difference between 220 and your age
Upper limit (bpm) = 75% of the difference between 220 and your age

- **What format are they in?**

Format floating-point: Double.

- **How do you output them? (e.g., on screen or to a file)**

The output of the value will be on screen.

Question 5b - Test Table - target heart rate
Create a test plan with sample data for the below problem. Make a table like this with enough rows. Recall that types of test cases to consider:

- for each different possible output
- for either side of a boundary, and on the boundary, implied in the problem
- for common known answers (like 32 F = 0 C)
- for low, average and high input values

At this time, we are not writing test cases for invalid input values. We are just identifying those in the Input-Process-Output design.

Table 5. Test table for problem 5

Test Case Reason	Sample Data		Expected Result (manually calculate)	
age = 0	age	0	lowthr	123.00
			highthr	153.75
age > 0	age	20	lowthr	102.00
			highthr	127.50

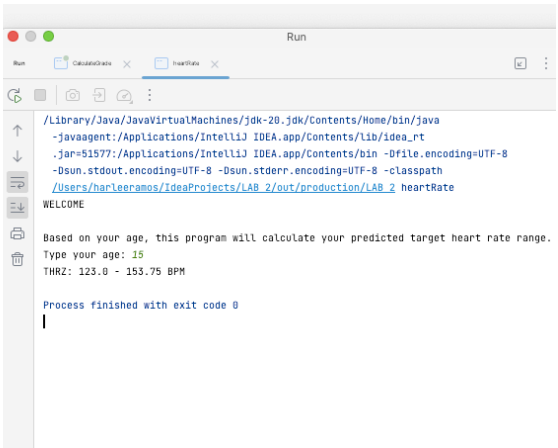
Question 5c - Write Code - target heart rate
Implement your Input-Process-Output Design in Java. Make sure to utilize these programming concepts correctly when appropriate:

- comments
- descriptive identifier names
- correct data types
- unit coversion
- String literals for output using `System.out.println()`
- constants
- order of operations
- mixed type arithmetic (implicit type casting)
- % operator (mod or remainder)
- explicit type casting

Since we have not yet covered user input, you are provided with starter code which will read user input from in the test cases provided. The starter code also provides the output statements for the test cases. Make sure to test your code on all the test cases provided.
EXPECTED OUTPUT

```
15
THRZ: 123.0 - 153.75 BPM
28
THRZ: 115.19999999999999 - 144.0 BPM
50
THRZ: 102.0 - 127.5 BPM
80
THRZ: 84.0 - 105.0 BPM
```

Figure 21. Problem 5: Test Case 1



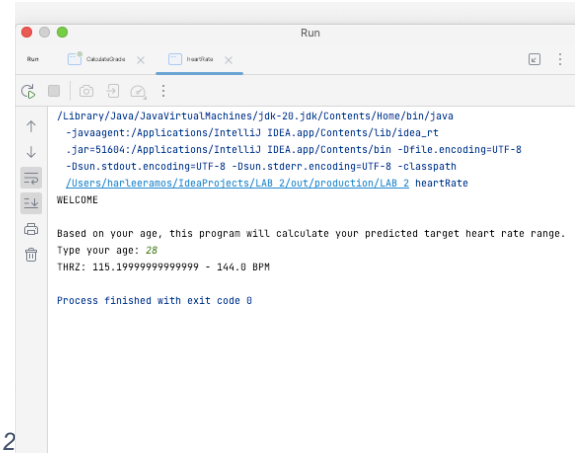


Figure 22. Problem 5: Test Case 2

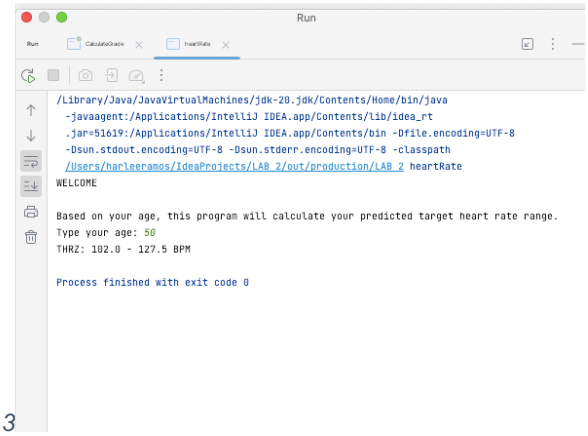


Figure 23. Problem 5: Test Case 3

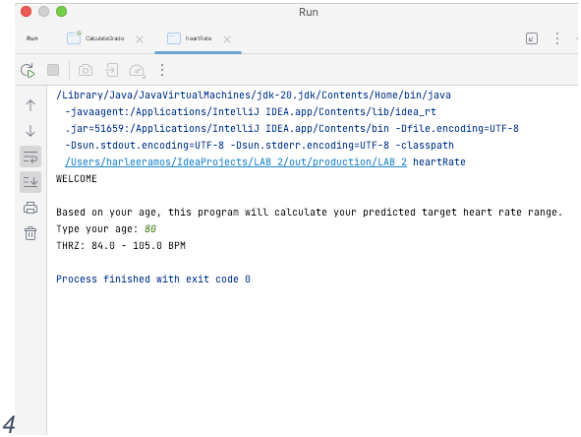


Figure 24. Problem 5: Test Case 4

QUESTIONS

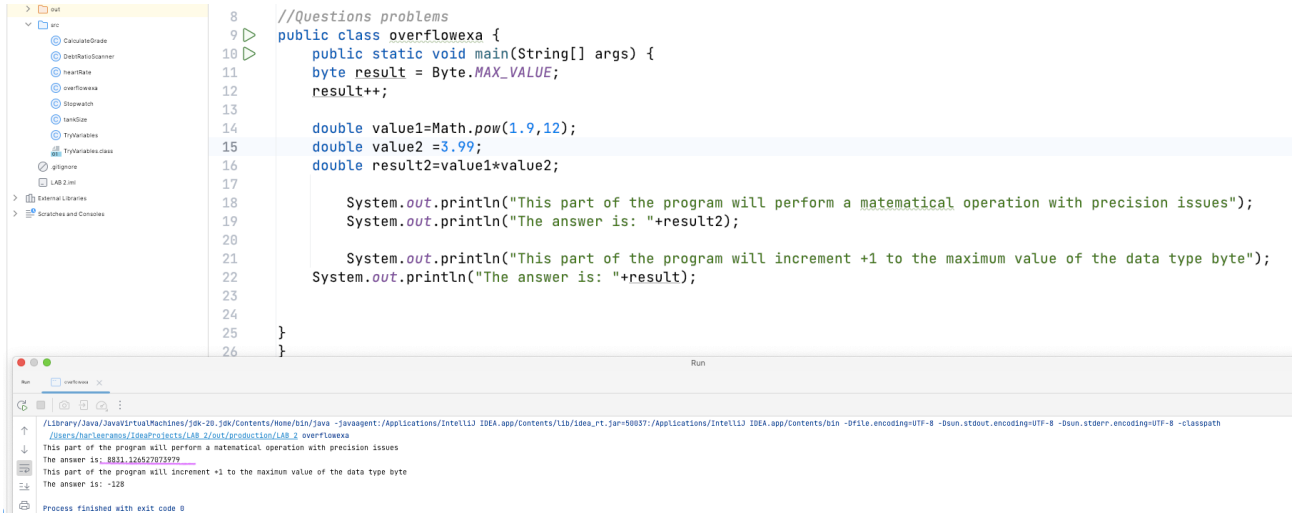
There are two issues with the accuracy of numeric computation with computers that we discussed. Please briefly explain the two issues, and how you would demonstrate these limitations with a few Java statements.

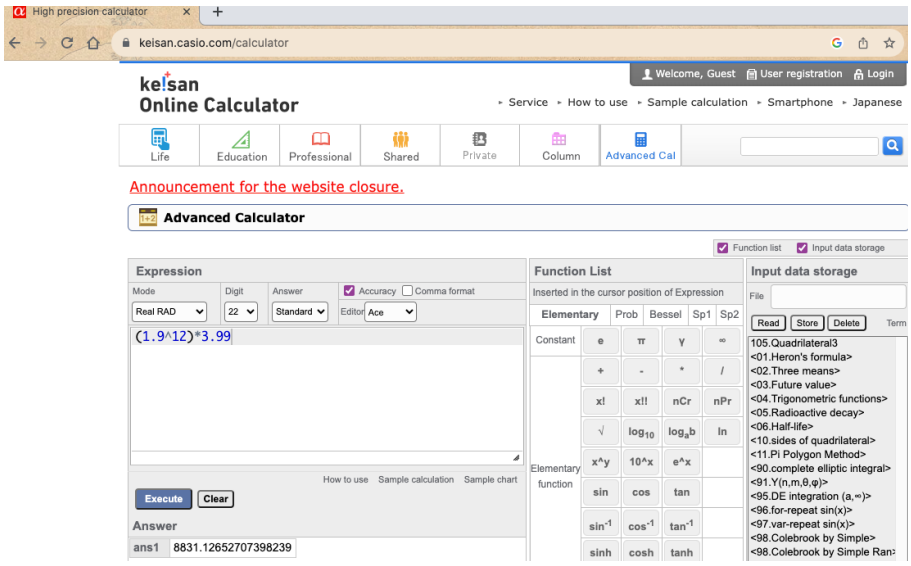
In class, we discussed that the accuracy of computers is finite due to the capacity of their storage. When we run operations with infinite decimals, Java will provide a rounded value that fixes its limits. In the java file called overflowexa, I added the example of accuracy. If we compare the output of the variable `result2` in Java with the online Kesian calculator of Casio on the website: https://keisan.casio.com/calculator, we compare the differences between their precision, see that Java rounds to many significant decimals.

JAVA OUTPUT
8831.12652707397**9**
KEISAN CASIO ONLINE HIGH PRECISION CALCULATOR
8831.126527073982**3**9

The second example could be the overflow of Java. With the example of incrementing the maximum value by default of a byte number in Java, the returned value will be the minimum value that the byte can store.

Both cases are depicted in the following picture:





In a few sentences explain what possible error issue integer arithmetic has in common with floating- point (real number) arithmetic and what error issue it does NOT have in common.

The overflow it's an issue that affects both data types. Because both were created with limits value of storage.

Precision issues are the main issue that affects the floating-point data type. Because Java was designed to provide only 15 digits of precision, if you perform an operation that exceeds that precision, the output number will be rounded to comply with Java limitations. If this occurs, the precision of the values will be lost due to it is off limits to represent it.

Table 2.1
Primitive Data Types

Data Type	Key Word	Cell Size (bytes)	Range and Precision
Integer Numeric	`byte`	1	-128 to +127
	`short`	2	-32,768 to +32,767
	`int`	4	-2,147,483,648 to +2,147,483,647
	`long`	8	-9,223, 372,036,854,775,808 to +9,223, 372,036,854,775,807
Real Numeric	`float`	4	±1.40129846432481707 E-45 to ±3.4028234663852886 E+38 *(7 digits of precision)*
	`double`	8	±4.94065645841246544 E-324 to ±1.797693134862157 E+308 *(15 digits of precision)*
Truth Value	`boolean`	1	`true` or `false`
One Character	`char`	2	Upper and lowercase keyboard characters and other entities (see Appendix C)