CS200: Programming I **Instructor:** Sam Vinitsky

P2: Doc Tok's Clock Shop

(20 points)

Due: Friday, June 28 @ 11:59pm

Part 0: Introduction

Brrrring brriiiiiing the phone rings – it's your old friend Doc Tok! Doc Tok informs you that his father recently retired, and that he has taken over operation of the family clock store. Doc Tok, who holds a Ph.D. in Clockology[1] says that he wants to be on the cutting edge of the clock sales business – as it is a very competitive field. Having heard that you recently enrolled in a computer science course, he has asked for your help in building state-of-the-art clock store technology.

He needs **three** programs written to help him run his clock store: (1) a program to help him paint the store, (2) a program to actually **run** the clocks, and (3) a program to help him handle his money. The programs are described in detail below, and each contains multiple functions. For each function, Doc Tok has given detailed specifications (input types and names, output types, special requirements, etc.) in a table below the function's description.

Doc Tok says that his computers are very slow, so you should only write functions that use simple arithmetic operations. **If we have not discussed it in class, his computer cannot run it.**[2] Specifically, his computers cannot run conditionals ("if" statements), loops ("for" or "while" statements), String parsing, or built-in Java libraries like Math or Random. If you have questions about what you are/not allowed to use, please ask him on Piazza.

Tip: Before attempting this homework, work through all of the practice problems from the lectures on functions (available on Canvas here).

Each program you turn in should include a comment at the top with (1) your full name, (2) your student ID number, (3) your netID, and (4) the name of anyone you discussed the homework with (excluding of Sam and Alex).

As always: **start early, ask questions, and have fun!**

---

[1] not a real field

[2] how convenient!

Part 1: Paint the store red [4 points total]

Doc Tok's first task for you is relatively simple – since his father had terrible taste, the current design of the clock store is very cliche. To help remedy this, Doc Tok has decided to paint the entire store with a new, bright color called "red"! But he needs your help figuring out how much paint to buy.

This seems like a simple task, but there's a catch – Doc Tok is very poor (as his Ph.D. is Clockology put him severely in debt). He cannot afford much paint, so he wants to paint as little of the store as he can. In particular, since the walls are covered in clocks, he can save money by not painting the parts of the wall that the clocks cover!

Doc Tok has asked you to write a program called `Paint.java`, and include three functions in it, which are described below.

You will need to test each function that you write to make sure it works. Follow the tips given in "Tips for Testing Functions".

Note: As discussed in class, `double`'s in Java are weird. Occasionally, you may notice that your `double`'s value is off by some small fraction – maybe your answer was supposed to be `26` but your program returned `26.000000000004`. This is due to the way that Java stores doubles internally, and is completely out of our control. Doc Tok will not care if your results are off by such a small amount.

a) Area of a clock (1 point)

In order to figure out how much of the wall is covered by clocks, Doc Tok realizes you must first have a way to find the area of a single clock. Write a function called `computeClockArea` that, given the radius of the clock as input, outputs the area of the clock (see the table below for more details).

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeClockArea`
Input(s):	`double radius`: the radius of the clock
Return value:	A `double` equal to the area of a clock with a radius of `radius`
Special requirements:	Pretend that π is just 3.14
Hints:	What is the formula for the area of a circle?
Example usage:	`computeClockArea(1.0)` should return `3.14` `computeClockArea(4.0)` should return `50.24`

b) Total wall area (1 point)

Next, Doc Tok needs to know just how big his walls are. Write a function called `computeWallArea` which, given the dimensions of a single wall, determines the total area of that wall. Notice that this function has **two** inputs!

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeWallArea`
Input(s):	`double width`: the width of the wall `double height`: the height of the wall
Return value:	A `double` equal to the surface area of the wall
Special requirements:	None
Hints:	What is the formula for the area of a rectangle?
Example usage:	`computeWallArea(2.0, 4.0)` should return `8.0` `computeWallArea(3.0, 3.0)` should return `9.0`

c) How much paint? (2 points)

You can now calculate how much paint you actually need to buy by calculating how much of the wall area is exposed. The key here is that any part of the wall that has a clock hanging on it doesn't need to be painted, since most of his customers don't have x-ray vision.

Since Doc Tok is very particular, all of the clocks are the exact same size, as are all the walls. To calculate how much of the walls are actually visible, you need to consider: (1) the number of clocks, (2) the radius of the clocks, (3) the height of the walls, (4) the width of the walls, and (5) the number of walls,

Write a function `computeExposedWallArea` that will return amount of paint that Doc Tok will need. Since your answer depends on five variables, your function will need **five** inputs. Some of these will need to be `double`'s, and some will need to be `int`'s. **Check the table below for the order of inputs, and their types**.

To write this function, you can (and should) use the functions you have written in parts a and b!

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeExposedWallArea`
Input(s):	`double radius`: the radius of each clock `int numClocks`: the number of clocks `double width`: the width of each wall `double height`: the height of each wall `int numWalls`: the number of walls
Return value:	A `double` equal to the exposed surface area of walls when `numClocks` clocks (each with a radius of `radius`) are hung on `numWalls` walls (each with a height of `height` and width of `width`)
Special requirements:	Call the functions you have written...
Hints:	Use variables to store intermediary values
Example usage:	`computeExposedWallArea(1.0, 2, 4.0, 4.0, 2)` returns `25.72`

Part 2: Making a clock [5 points total]

Doc Tok is sick of his father's old, antiquated clocks. He wants to bring in some fancy new electronic clocks, but needs your help writing the code to run them! Specifically, your job will be to write code to determine the **angle of the hands of the clock**. If you don't know how to read an analog clock (or just need a refresher) check out this page. Doc Tok uses military time, so his hours go from 0 to 23.

In a real clock, the hour hand is dependant on the current hour, minute, and second (since when it's 11:59:59, the hour hand should be *almost* at the 12). Similarly, the minute hand is dependent on the current second. Doc Tok knows you are just learning how to program, so he does not expect you to implement those interactions.

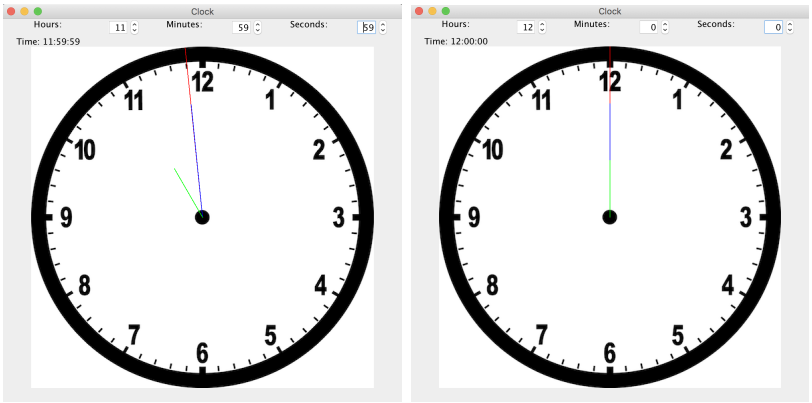

Figure 1: Clocks at 11:59:59 (left) and 12:00:00 (right)

Instead, the hour hand of your clock will be dependent **only** on the current hour. At 11:59:59, your hour hand (the short green one) should still be at 11. Then at 12:00:00, it should jump to

the 12 (see Figure 1). Similarly, the minute hand (the medium length blue one) should show the current minute, without regard to how many seconds have passed. At 4:30:59, it should still be at the "30 minute mark" (which is pointing directly down, at the 6). At 4:31:00, it should move to the tick immediately following the 6 (**not** the 7) (see Figure 2). This simplification does not affect the seconds hand (the long red one).

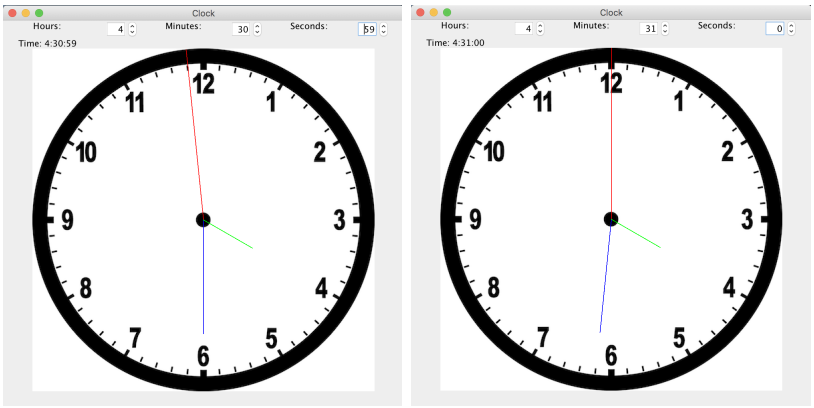

Figure 2: Clocks at 4:30:59 (left) and 4:31:00 (right)

Write a program called `Clock.java` with the following functions:

```
int computeSecondHandAngle(int seconds)
int computeMinuteHandAngle(int minutes)
int computeHourHandAngle(int hours)
```

See the tables below for a more detailed description of the functions.

Framework

Doc Tok has written a program called `ClockFramework.java` that will display a clock by running your functions to calculate the angles of the hands. This means that if there are errors in your code, they will be reflected in the operation of this clock.

In order to run the framework, make sure it is in the same folder as `Clock.java`. Open up `ClockFramework.java` in Visual Studio Code, and run it (using the "run" button in the top-right, as usual). If you have written your functions in `Clock.java` (and named them correctly!), it should display a clock that you can fiddle with to test your code.

You can adjust the time displayed on the clock by using the selectors at the top of the interface. For more information about how to use this, read the long comment at the beginning of `ClockFramework.java`.

Note: You are welcome to read the code of `ClockFramework.java`, but I would recommend against it, as it contains complicated graphics functions and is far beyond the scope of anything we have done so far, and it would probably just scare you at this point.

a) Second Hand (2 points)

Function Name:	`computeSecondHandAngle`
Input(s):	`int seconds`: between 0 and 59 (inclusive)
Return value:	`int` representing the angle (in degrees) of the second hand when the current time has `seconds` seconds passed
Special requirements:	0 degrees is straight up (12 o'clock) Degrees increase counterclockwise (3 o'clock is 90 degrees) You may **not** use "if" statements. The angle returned must be between 0 and 360
Hints:	Remember int division... Percentages

Example usage:	`computeSecondHandAngle(0)` should return `0`
	`computeSecondHandAngle(1)` should return `6`
	`computeSecondHandAngle(10)` should return `60`
	`computeSecondHandAngle(30)` should return `180`

b) Minute Hand (1 point)

Function Name:	`computeMinuteHandAngle`
Input(s):	`int minutes`: between 0 and 59 (inclusive)
Return value:	`int` representing the angle (in degrees) of the minute hand when the current time as `minutes` minutes, **ignoring the contributions from the seconds**
Special requirements:	0 degrees is straight up (12 o'clock) Degrees increase counterclockwise (3 o'clock is 90 degrees) You may **not** use "if" statements. The angle returned must be between 0 and 360
Hints:	This should look familiar...

Example usage:	`computeMinuteHandAngle(0)` should return `0`
	`computeMinuteHandAngle(1)` should return `6`
	`computeMinuteHandAngle(10)` should return `60`
	`computeMinuteHandAngle(30)` should return `180`

c) Hour Hand (2 points)

Function Name:	`computeHourHandAngle`
Input(s):	`int hours`: between 0 and 23 (inclusive)
Return value:	`int` representing the angle (in degrees) of the hour hand when the current time has `hours` hours **ignoring the contributions from minutes and seconds**
Special requirements:	0 degrees is straight up (12 o'clock) Degrees increase counterclockwise (3 o'clock is 90 degrees) You may **not** use "if" statements. The angle returned must be between 0 and 360
Hints:	A clock only goes up to 12, but military time foes from 0 to 23...

Example usage:	`computeHourHandAngle(0)` should return `0`
	`computeHourHandAngle(2)` should return `60`
	`computeHourHandAngle(6)` should return `180`
	`computeHourHandAngle(18)` should return `180`

Part 3: Money money money... [8 points total]

As mentioned before, Doc Tok is quite frugal. He has asked that you write him a program called `Money.java` to help him make sure he isn't making any mistakes in his financial transactions.

This program will include three functions, specified below.

a) Paying employees (2 points)

Doc Tok's father was over-paying his employees for years, since he was didn't know how to read or write, and didn't believe in taxes. Doc Tok hopes to remedy this oversight by using computers to pay his employees!

Each employee makes a certain amount of money per hour, and works a certain number of hours per pay period. Additionally, some percentage of their paycheck gets taken out for taxes. Write a function called `computeEmployeePaycheck` to help figure out how much Doc Tok needs to actually pay his employees (details in the table below).

For example, suppose an employee works `10` hours at a salary of `5.00` dollars an hours and a tax rate of `7`%. The total amount they make (before tax) is `10*5.00`, which is equal to `50.00` dollars. Then `7`% of this is removed for tax – i.e. the amount removed for tax is `0.07 * 50.00`, which is equal to `3.5`. So the amount the employee actually gets paid is `50.00-3.50` which is `46.50` dollars.

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeEmployeePaycheck`
Input(s):	`int numHours`: number of hours they should be paid for `double salary`: amount to pay per hour (in dollars) `int tax`: percentage of their salary that goes to tax
Return value:	A `double` equal to amount the employee should be paid based on the inputs.
Special requirements:	None
Hints:	`tax` is a whole number, but you need it as a decimal... Do you remember how to convert from an `int` to a `double`?
Example usage:	`computeEmployeePaycheck(10, 10.0, 4)` should return `96.0` `computeEmployeePaycheck(20, 8.5, 8)` should return `156.4`

b) Making change (4 points)

When making change for a customer, Doc Tok likes to use as few coins as possible (since Clockology believes that all circles are precious, as they closely resemble clocks). Doc Tok would like you to write a function that will make change for a given amount of money in as few coins as possible. He wants the function to both (a) **print out** the number of coins he will be needing of each type, and (b) **return** the total number of coins he will need. The input to this function will be an `int` representing the number of **cents** to make change for. (see the table below for more details)

What you print should follow this format:

```
Total: 15
Quarters: 3
Dimes: 0
Nickels: 10
Pennies: 2
```

He will only give change in basic American coins, which are the quarter (worth 25 cents), the dime (worth 10 cents), the nickel (worth 5 cents) and the penny (worth 1 cent).

For example, consider making change for `144` cents. First, we want to try to use as many quarters as possible, each of which are worth 25 cents. How many times does `25` go into `144`? We can cleanly subtract it `5` times, which amounts to `125` cents (so the number of quarters we give is `5`). The remainder is `19` cents. Now we move to the next largest coin, the dime (worth 10 cents). We can only subtract out `1` dime, which gives a remainder of `9` cents. Then we can subtract out a single nickel (worth `5`), giving a remainder of `4` cents. We then give out `4` pennies, since this is the smallest coin. The total number of coins we need is `5+1+1+4`, which is equal to `14`.

We would then print:

```
Total: 11
Quarters: 5
Dimes: 1
Nickels: 1
Pennies: 4
```

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeChangeWithFewestCoins`
Input(s):	`int amount`: amount of money to make change for (in cents)
Return value:	`int` number of coins needed
Print out:	The number of each coin type needed to make change for `amount` using the *fewest* number of coins possible (following the format mentioned above)
Special requirements:	Coins are: Quarter (25 cents), dime (10 cents), nickel (5 cents), penny (1 cent)
Hints:	- Make an `int` variable called `amountLeftToMakeChangeFor` or something like that, and update it as you go... - Might want variables `numQuarters`, `numNickels`, etc... - We need to compute the number of times that a value can be cleanly subtracted from an amount, and then the remainder... - What do % and `/` do?

Example usage:	`computeChangeWithFewestCoins(51)` should return `3`
	`computeChangeWithFewestCoins(141)` should return `8`
	`computeChangeWithFewestCoins(408)` should return `20`
	`computeChangeWithFewestCoins(4)` should return `4`

c) Clock sales (2 points)

When a customer comes in and makes a purchase, they buy a bunch of the same type of clock, and pay him in a pile of assorted coins. (Doc Tok has a very particular clientele). Doc Tok then needs to compute the amount they over-paid by, make change for it, then give that amount back to them. As discussed above, Doc Tok enjoys doing this with as few coins as possible. Using the change-making function you just wrote, write a function to compute how much change Doc Tok needs to give a customer. (see the table below for more details)

Make sure to test your function as described in "Tips for Testing Functions".

Function Name:	`computeChangeFromSale`
Input(s):	`int numClocks`: number of clocks sold `int clockPrice`: price of each clock (in cents) `int amountPaid`: amount of money the customer paid (in cents)
Return value:	`int` number of coins needed
Print out:	The coins to give for the customer's change from the sale.
Special requirements:	Use the function `computeChangeWithFewestCoins` in your code.
Hints:	Which amount are you making change for here?

Example usage:	`computeChangeFromSale(2, 250, 600)` should return `4`
	`computeChangeFromSale(2, 250, 500)` should return `0`
	`computeChangeFromSale(3, 210, 1000)` should return `16`

Part ∞: Feedback Form [3 points]

Fill out this feedback form **after you submit** this assignment. Completion of this will count towards your grade, but your responses themselves will not affect your grade in any way (so be honest!).

What to turn in

On Canvas, turn in a **zip** folder named `<your_net_id>_P2.zip` containing the files:

- `Paint.java` [4 points]
- `Clock.java` [5 points]
- `Money.java` [8 points]

It is fine if you also include your `.class` files and `ClockFramework.java`, but this is not required (it may just be easier to include them than to delete the files in order to zip up your folder).

Make sure to complete the feedback form as well! [3 points]
