Project - Part 02

Important

This part of the assignment is an extension to the original Part 01 document. **Please make sure you submit to Part 01 before modifying it as specified below.**

Step 01 - InvalidSinExceptionClass (/2 marks)

In this step, you will start by building a new programmer-defined exception class called **InvalidSINException**. This exception class will be thrown whenever a SIN number provided to a constructor or setter is not valid. We will talk about what makes a SIN valid later in the document.

For now, you are to create an **InvalidSIN.java** file which extends the Exception class and consists of:

1) No-argument constructor
   - This constructor should set the default message to "ERROR: Invalid SIN number provided"
2) One argument constructor taking a single String value
   - This constructor should set the message to the String parameter value provided to the constructor

Step 02 – Update setSIN method(/5 marks)

In part 01 of the project that you have already submitted you were asked to develop a *setSIN* method. Initially, I had no validation requirements; however, SIN numbers can be validated quite easily following the below steps:

1. Start with a SIN number you would like to verify -→ i.e. 046 454 286
2. Start by ensuring SIN number is not null, consists of only numeric values, and is of length 9
   a. If not, you should throw a default InvalidSINException
3. Extract the check digit (9th digit) → 046 454 28 **6**
4. Double each even placed digit → 0**4**6 **5**4 2**8** (8, 8, 8, 16)
5. Add the digits from step 04 together → 8 + 8 + 8 + 1 + 6 = 31
6. Add the odd placed digits → **0**4**6** 4**5**4 **2**8 (0 + 6 + 5 + 2 = 13)
7. Add the results of step 5 and 6 together → 31 + 13 = 44
8. Round result of step 07 up to the nearest multiple of 10 → 50
9. Subtract the result of step 7 from step 8 → 50 – 44 = **6**
10. If the result od step 09 matches the check digit (9th SIN digit) then it is a valid SIN.
    a. If the SIN is not valid, throw a default InvalidSINException

Step 03 – Remaining Fields (/2 marks)

In this step, you will be asked to throw an **InvalidArgumentException** for the situations listed in table below:

method	When to Throw	Message
setFirstName()	➢ null parameter provided ➢ empty string for name	"Error: Invalid first name provided"
setLastName()	➢ null parameter provided ➢ empty string for name	"Error: Invalid last name provided"
setSalary()	➢ value of less than 0	"Error: weekly salary cannot be a negative value"
setHours	➢ value less than 0	"Error: hours worked cannot be a negative value"
setHourlyRate	➢ value less than $14.10 (this is student's minimum wage in Ontario)	"Error: hourly rate of pay must be minimum wage or higher"
setComissionRate()	➢ value less than 0% ➢ value greater than 100%	"Error: commission rate should be between 0% and 100%"
setGrossSales	➢ value less than 0	"Error: gross sales must be a value of 0 or greater"

Note: Your constructors, if written properly, should call these methods either directly or indirectly (through super()) and therefore will also throw these errors if invalid values are provided.

Step 04 – Correct Employee SIN numbers (/1 marks)

In part 02 of the project you were asked to construct 4 employees objects: Joe Francis, Samantha Hughes, Kim Adams, and Ryan Goodall.

These employees were given SIN numbers; however, they may not be valid. Please correct these SIN numbers to valid values. The rest of the test program should remain unchanged.

Step 05 – Create EmployeeExceptionTester.java (/3 marks)

The final step of the project is to create a driver file to verify your exceptions. This file should allow the user to enter one employee of any type and then provided the information needed to construct it. An example for salaried Employee is shown below.

```
Welcome to Employee Exception Tester!

Which of the following types of employees would you like to create:
   1. Salaried
   2. Hourly
   3. Commission

Please enter your selection: 1

Please enter employee's first name: Aaron
Please enter employee's last name: Sarson
Please enter employee's SIN: 00000000
Please enter employee's weekly salary: $500
**********************************************
```

ERROR: Invalid Social Insurance Number
You have entered incorrect information. We will now re-prompt you for all inputs.

Please enter employee's first name: **Aaron**
Please enter employee's last name: **Sarson**
Please enter employee's SIN: **046454286**
Please enter employee's weekly salary: $**-300**
****

Error: Weekly salary cannot be a negative value
You have entered incorrect information. We will now re-prompt you for all inputs.

Please enter employee's first name: **Aaron**
Please enter employee's last name: **Sarson**
Please enter employee's SIN: **046454286**
Please enter employee's weekly salary: $**500**
****

Employee successfully created!

Type: Salaried
Name: Sarson, Aaron
SIN: 046454286
Weekly Salary: $500

Guidance for Step 05:

- The initial reading of the employee types the user would like to create should be done outside of the main looping structure.
    - You should ensure a selection is between 1 and 3
        - If not re-prompt the user until a valid entry is provided
    - In the event of an *InputMismatchException* catch the exception and display *getMessage()* to the console.
        - Your program should then re-prompt the user for a valid value between 1 and 3

- After a correct employee type selection is made, a value between 1 and 3, use a decision structure of your choice to read in the required information for that type of employee
    - Once all data is entered, call the appropriate constructor. These constructors should call the methods which will throw the exceptions.
    - Regardless of the exception thrown, display *getMessage()* followed by the String "You have entered incorrect information. We will now re-prompt you for all inputs."
        - The code we wrote ourselves could throw either an **InvalidSINException** or **InvalidArgumentException**; however, remember the Scanner could throw an **InputMismatchException**.
            - All 3 of these exceptions should be handled as mentioned above.

- If an exception is thrown, re-prompt the user for all the information again
  - Repeat doing this until no exceptions are thrown
- Once the Employee object is created successfully exit the loop and display “Employee successfully created!”
  - Finally, call the object’s toString() method and display its contents.