CSE 1322L – Assignment 6 (Fall 2025)

Introduction

Of the many departments that the Georgia Bureau of Investigations has, one of them is the Chemistry Section. Its primary duty is to analyze suspected drugs and fire debris collected by law enforcement officers in order to identify its composition. Once analyzed, the results are sent back to aid in their investigations, potentially being used as evidence in a court case down the line.

In this assignment, we will write a program which reads in information from such analysis, evaluates if the material is a controlled substance and, based on Georgia Law, determines what penalty should be applied to the culprit. Unfortunately, while GBI's Chemistry Section publishes which analytical examinations it performs, there are no technical details as to how they are conducted or what the expected results are. As such, we will have to be creative: we'll incorporate some of the provisions under OCGA 16-13-26, 16-13-30, 16-13-31, (which deal with the possession and trafficking of controlled substances), and we'll reference some of the tests the Chemistry Section uses by name; we'll make up everything else.

Drug Tests

Our pretend lab will perform tests on 3 different types of drugs: marijuana, cocaine, and methamphetamine. There is a total of 5 different tests that can be run on any given drug, with the first two tests being common to all drugs, and the third test differing between all drugs. The table below shows which tests are run on which drugs.

Drug type	Gas Chromatography	Mass Spectrometry	Gas Chromatography Abundance	Ultraviolet Spectroscopy	Logo ID
Marijuana	✓	✓	✓		
Cocaine	✓	✓		✓	
Methamphetamine	✓	✓			✓

Once a drug is sampled, its results are entered into a comma-separated file which has the following format:

```
drug_type,weight,test_1,test_2,test_3
```

The tests in the file are determined by the drug_type and follow the order that they appear in the table above. Thus, when reading a marijuana sample, test_1 is a Gas Chromatography test, test_2 is a Mass_Spectrometry test, and test_3 is a Gas Chromatography Abundance test.

Your program will not produce this file; it will read from it. Each line will contain one drug being sampled which, going forward, will be referred to as “samples”. Your program will read a sample’s information, perform tests on it as per the order that they appear in the table above, and then write 2 files which will contain the report results.

Below you can find a description of each test, as well as the expected results for each drug. **For a sample to be confirmed to be a drug, it must pass all three of its tests**.

Gas Chromatography

This test result will be a positive integer, which indicates how many seconds a drug was in the Gas Chromatographer before the machine could separate and detect its components. The table below shows the separation time of each drug.

Drug	Time
Marijuana	5:47 - 6:14
Cocaine	6:38 - 7:02
Methamphetamine	5:07 - 5:16

Both endpoints are inclusive: if a cocaine sample was in the machine for 6 minutes and 38 seconds, or for 7 minutes and 2 seconds, or anything in between, it “passed” the test (i.e.: it could be cocaine).

Mass Spectrometry

This test result will be 3 positive integers, separated by spaces, which indicate the weight of the heaviest pieces of the sample (in m/z), herein referred to as “peaks”. For a sample to “pass” the test and potentially be considered a drug, at least 2 of its peaks must match the values in the table below.

Drug	Peaks
Marijuana	`314 299 231`
Cocaine	`149 91 58`
Methamphetamine	`303 182 82`

A methamphetamine sample which has “`303 100 82`” as its peaks could potentially be methamphetamine as both 303 and 82 match the values on the table. However, a sample with peaks of “`303 181 83`” fails the test as it only has 1 match.

Gas Chromatography Abundance

During this test, the machine compares the concentration of the sample against a standard sample whose drug concentration is known. This allows the machine to derive the concentration of the sample under testing. This test result will be a floating-point number. Values 0.3 and above pass the test, whereas any other value fails it.

Ultraviolet Spectroscopy

This test shoots ultraviolet light at the sample and then measures which wavelength is absorbed by the sample, within a margin of error. This test result will be an integer. Values below 192 or above 202 fail the test.

Logo ID

All medication produced in our fictional world has an engraving, a specific color, and shape, all of which is registered in a government database when the drug is approved for sale. For this test, our lab looks at the drug under a microscope and checks the drug's characteristics against the database, writing down the results.

This test result will be 3 strings, separated by spaces. The first string is the engraving on the medication, the second is its shape, and the third is its color. Only the following variants of Desoxyn are legal (**thus "failing" the test**). All other variants pass the test.

Engraving	Shape	Color
R-12	round	white
V-20	oval	blue
A-65	capsule	pink

Penalties

Once a sample is confirmed to be the drug that it is being tested for, your program must then determine the penalty according to the law. The only factor which will influence the penalty is the weight of the sample, which is stored in the file as a floating-point number. The tables below show the penalties.

Cocaine and Methamphetamine	
Weight (grams)	**Penalty**
Less than 1g	Up to 3 years in prison
1g or more but less than 4g	1 - 8 years in prison
4g or more but less than 28g	1 - 15 years in prison
28g or more but less than 200g	Minimum 10 years, $200,000 fine
200g or more but less than 400g	Minimum 15 years, $300,000 fine
400g or more	Minimum 25 years, $1,000,000 fine

Marijuana	
Weight (grams)	**Penalty**
Less than 28.35g	Misdemeanor
28.35g or more but less than 4535g	1 - 10 years in prison
4535g or more but less than 907184.7g	Minimum 5 years, $100,00 fine
907184.7g or more but less than 4535924g	Minimum 7 years, $250,000 fine
4535924g or more	Minimum 15 years, $1,000,000 fine

Note: marijuana is usually measured in ounces and pounds. Here, those have been converted to grams for consistency.

Requirements

The features described below must be in your program:

- A class Sample
    - Has the following fields:
        - An integer called “id”
        - A static integer called “nextId”, initialized at 0
        - A String called “drugType”
        - A double called “weight”
        - An integer called “gcTime”
        - A String called “msPeaks”
        - A String called “miscTest”
        - An arraylist of Strings called “tests”
    - **Sample(String, double, int, String, String)**: Initializes “tests”, assigns “nextId” to “id” then increments “nextId” by 1. Finally, it utilizes the arguments to initialize the rest of the fields, in the order that they appear above.
    - All fields have a getter except for “nextId”
    - **String toString()**: this override returns the following String (replace the curly braces and their content with the value in the object’s fields):

        ```
        Case: #{id}
        Drug: {drugType}
        Weight: {weight}g
        Tests run: {tests}
        ```

        `{tests}` should be comprised of the tests inside the “tests” field, separated by a comma and a space. Thus, if a marijuana sample with “id” 10, weight 200, and which has “Gas Chromatography” and “Mass Spectrometry” in its “test” field were to be printed out, its output would be as follows:

        ```
        Case: #10
        Drug: Marijuana
        Weight: 200.00g
        Tests run: Gas Chromatography, Mass Spectrometry
        ```

- A class TestFailedException
    - Must be a subclass of Exception
    - Has a single field of type Sample called “sample”
    - **TestFailedException(String, Sample)**: Passes the string to its superclass constructor and assigns the Sample to its own field
    - “sample” has a getter
- Your driver class must have the following static methods:
    - **void processMarijuana(Sample)**: This method evaluates the Sample as if it were marijuana, executing the tests as outlined under Drug Tests. As tests

are performed, append them to the Sample's "tests" field. If a sample fails a test, throw a TestFailedException. This method does not catch any exceptions and, as such, **must have no TRY-CATCH blocks in its body**. Below you can find the messages that must be passed to the exception if a sample fails a test; pass only the message which reflects which test failed:

- "Separation time out of bounds"
- "Insufficient peak matches"
- "Concentration below 0.3%"

- **void processCocaine(Sample)**: This method evaluates the Sample as if it were cocaine, behaving like processMarijuana() but for cocaine. Exceptions thrown by this method must have one of the following messages:
  - "Separation time out of bounds"
  - "Insufficient peak matches"
  - "UV reading out of range"
- **void processMethamphetamine(Sample)**: This method evaluates the Sample as if it were methamphetamine, behaving like processMarijuana() but for methamphetamine. Exceptions thrown by this method must have one of the following messages:
  - "Separation time out of bounds"
  - "Insufficient peak matches"
  - "Prescription medication"
- **void processFile(Scanner)**: The Scanner passed to this method will read from a file containing drug samples, as outlined under Drug Tests. Samples which pass the test are confirmed to be illegal drugs, being then written to a file. Samples which fail a test will be written to a different file.
  - Read a line from the file
  - Determine which drug is being evaluated
  - Create a Sample object with that line's information
  - Pass it to the appropriate method (one of the three above)
  - If the method throws an exception, write the following to "failed.txt":

```
Case: #{id}
Drug: {drugType}
Weight: {weight}g
Tests run: {tests}
Result: Negative, {exception message}
====
```

  - If the method does not throw an exception, write the following to "passed.txt" (where "{penalty}" is the appropriate penalty as outlined under Penalties):

```
Case: #{id}
```

```
Drug: {drugType}
Weight: {weight}g
Tests run: {tests}
Result: Positive, {penalty}
====
```

        - Repeat the above steps until the Scanner runs out of lines to read

    - **void main()**
        - Prompts the user for a file name
        - Creates a File object and creates a Scanner with it
        - If the file does not exist, print out an error message then terminate the program
        - If the file exists, pass it to processFile()
        - Once processFile() is done executing, print out a success message, then terminate the program

Deliverables

- Assignment6.java (driver)
- Sample.java
- TestFailedException.java

Considerations

- You will get partial credit for partial work, as long as the rubric permits it.
- Despite what deliverables say, all of your classes can be submitted in a single file.
- You will have to make use of String methods to extract the information you need from the input file. You can find its documentation here.
- Do not forget to flush your output streams and to close your streams at the end!
- While the slides recommend that you close your streams using the FINALLY in a TRY-CATCH-FINALLY block, feel free to use TRY-WITH-RESOURCES if you know how.
- Most of the input fed into this program and the output generated by this program is present in files not contained in this write-up
    - Be sure to download the sample input and sample outputs from the website
    - You can double-check if your outputs match the ones provided by running them through your Lab 9
    - Alternatively, you can use this site
    - All data in the sample input was randomly generated. Any matches to real world cases are purely coincidental
- If you would like a different input file to test your program (with its matching output), you can request a different one once via email
    - You can select how many samples the file contains; minimum of 500, maximum of 1000000

Sample Output (user input in red)

```
[Drug Report Analyzer]
Enter name of drug file: drugs.txt
Could not find file 'drugs.txt'

Program complete.
```

Sample Output (user input in red)

```
[Drug Report Analyzer]
Enter name of drug file: samples.txt
File loaded, processing...
File processed. Outputs written to 'passed.txt' and 'failed.txt'.

Program complete.
```