Concordia University
COMP 249 – Winter 2025
Object-Oriented Programming II - Assignment # 3
Due 11:59 PM – Friday, April 11th, 2025

Purpose: The purpose of this assignment is to allow you to practice File I/O, ArrayList, and Linked Lists.

IMPORTANT NOTE: IN THIS ASSIGNMENT, YOU ARE **NOT PERMITTED** TO USE ANY OF THE JAVA BUILT-IN CLASSES SUCH AS LinkedList, HashMaps, *etc.* Using any of these will result in zero marks! In other words, you need to code whatever is needed by yourself!!! Of course, you can use the classes String, Scanner and ArrayList.

General Guidelines When Writing Programs:

- Include the following comments at the top of your source codes

```
// -------------------------------------------------------
// Assignment (include number)
// Question: (include question/part number, if applicable)
// Written by: (include your name and student id)
// -------------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message which includes your name(s).
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy-to-read format.
- End your program with a closing message so that the user knows that the program has terminated.

"A tariff war is like setting fire to your own house to smoke out your neighbor."

— ***Warren Buffett***

Trade has long been the lifeblood of nations, fueling economies, shaping alliances, and sometimes, igniting conflicts. In a world where globalization once promised seamless commerce, nations now find themselves entangled in a new kind of battle—Tariff Wars. Countries impose tariffs not just as economic policies, but as weapons in strategic confrontations. A simple percentage increase on imported steel can ripple through industries, driving up costs, shifting jobs, and altering global supply chains. What begins as a minor trade dispute can escalate into a full-scale economic standoff, leaving consumers, businesses, and even entire nations caught in the crossfire. With tensions rising between economic superpowers, new tariffs are being imposed, retaliatory measures are being enacted, and the global marketplace is becoming increasingly unstable. As key players in this geopolitical chess game, policymakers must decide which industries to protect, which alliances to maintain, and how to prevent economic collapse.

As an economic strategist, your task is to design a Tariff Management System that simulates the effects of international trade policies. You will:

- ✓ Analyze and adjust tariffs on various goods using File I/O & ArrayLists.
- ✓ Track and manage trade disputes using Linked Lists.
- ✓ Develop a system that allows adding, removing, and resolving trade conflicts.
- ✓ Simulate how tariffs impact global trade dynamics.

Part 1: ArrayList & File I/O

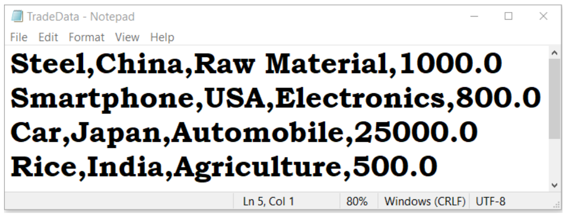
You are required to implement a program that reads trade data from a text file (*e.g.*, TradeData.txt), applies tariff increases based on specific rules, and writes the updated trade data to another text file (*e.g.*, UpdatedTradeData.txt). The program must handle the following operations using ArrayLists and File I/O.

Your implementation should be efficient, ensuring the smooth processing of large datasets with thousands of trade records.

Input File Format:

The program will read a list of products with their corresponding details, from a text file. Each line in the file represents a product entry, formatted as follows:

<ProductName>,<Country>,<Category>,<InitialPrice>



```
Steel,China,Raw Material,1000.0
Smartphone,USA,Electronics,800.0
Car,Japan,Automobile,25000.0
Rice,India,Agriculture,500.0
```

Figure 1. Sample TradeData.txt file

Tariff Adjustment Rules:

For each product, the tariff is applied based on the country of origin. The tariffs should be added to the product's initial price according to the following rules:

Country	**Tariff Increase**	**Affected Categories**
China	+25%	All products
USA	+10%	Electronics
Japan	+15%	Automobiles
India	+5%	Agriculture
South Korea	+8%	Electronics
Saudi Arabia	+12%	Energy
Germany	+6%	Manufacturing
Bangladesh	+4%	Textile
Brazil	+9%	Agriculture

Table 1. Tariff adjustment rules

Program Requirements:

1. **Read Data from a File:**

   Your program should read the trade data from the provided input file TradeData.txt. The data should be stored in an **ArrayList** of Product objects.

2. **Apply Tariffs:**

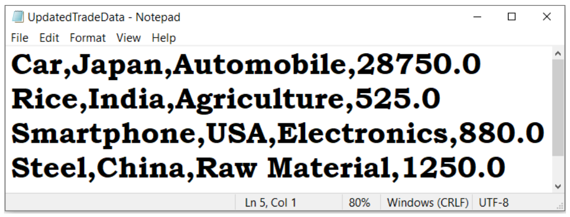
   Based on the product's country and category, the tariff rules should be applied to adjust the price of the products.

3. **Sort Products:**

   After applying the tariff adjustments, the product list should be sorted **alphabetically** by the product name.

4. **Write Data to a File:**

   The updated product data (with applied tariffs) will be written to file, UpdatedTradeData.txt, in the same format as the input file. Output file corresponding to Figure 1 is depicted in Figure 2.



```
Car,Japan,Automobile,28750.0
Rice,India,Agriculture,525.0
Smartphone,USA,Electronics,880.0
Steel,China,Raw Material,1250.0
```

Figure 2. Sample UpdatedTradeData.txt file

Here are the most important technical rules that you need to follow:

- You **MUST** use the **ArrayList** class to implement what is needed. In specific, when you read the input file, all data must be stored in one, or more, ArrayList objects.
- Your program should work for any text file name TradeData.txt.
- You **MUST NOT** use any other built-in Java classes/packages to do what is needed. For instance, you are **NOT** allowed to use other classes such as List, Map, HashMap, etc.
- In fact, all the needed imports for your program would be from java.io and java.util

Part 2: Linked Lists

Tariff regulations significantly impact commerce between nations, determining the viability of trade agreements and economic interactions. To navigate these complexities, an efficient system for managing trade tariffs is essential. In this assignment, you will design and implement a **Tariff Management System** that simulates trade conflicts between countries using linked lists. The system will process tariff data from input files, determine how tariffs affect trade between countries, and produce an output file summarizing the imposed tariffs and their effects.

You are given two files:

- **Tariffs.txt**: Contains information about imposed tariffs and minimum acceptable tariffs set by the destination country, categorized by product type.
- **TradeRequests.txt**: Contains trade requests between countries, including proposed tariff rates and product categories.

Your program will parse these files and generate an outcome for each trade request, considering the tariff restrictions in place.

Functionality Overview

For each trade request, your program should determine and output one of the following outcomes:

- ***Trade request accepted***: The proposed tariff meets or exceeds the minimum required tariff.
- ***Trade request conditionally accepted***: The proposed tariff is lower than the minimum but within 20% of it. A surcharge (formulation mentioned below) is applied to make up for the shortfall.

  $$\text{Surcharge} = \text{Trade Value} \times ((\text{Minimum Tariff} - \text{Proposed Tariff}) / 100)$$

- ***Trade request rejected***: The proposed tariff is more than 20% below the required tariff.

Your system must ensure that:

- Tariff rates are applied correctly based on trade relations as well as product categories.
- Trade requests are processed efficiently, ensuring each country's trade regulations are adhered to.
- Output is generated in a structured format, listing accepted, conditionally accepted, and rejected trades with specifics about every decision.

File Formats

Tariffs.txt

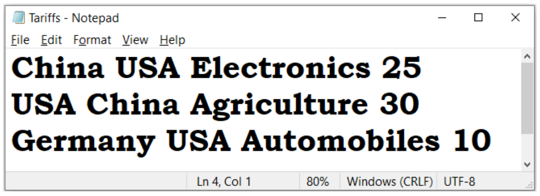
This file contains tariff details between multiple countries for different product categories. Each entry consists of:

DestinationCountry OriginCountry ProductCategory MinimumTariff

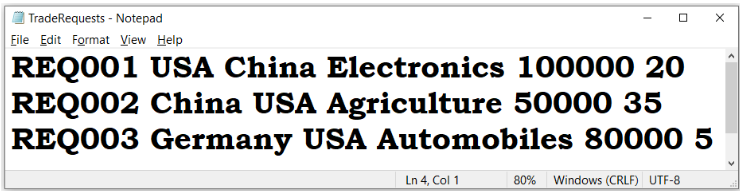
TradeRequests.txt

Each trade request consists of:

RequestID OriginCountry DestinationCountry ProductCategory TradeValue ProposedTariff



```
China USA Electronics 25
USA China Agriculture 30
Germany USA Automobiles 10
```



```
REQ001 USA China Electronics 100000 20
REQ002 China USA Agriculture 50000 35
REQ003 Germany USA Automobiles 80000 5
```

(a) (b)

Figure 3. Sample files (a) Tariffs.txt and (b) TradeRequests.txt

Implementation Details

I) Create an interface named **TariffPolicy** which has a method evaluateTrade that determines the trade request outcome based on tariff policies.

```
String evaluateTrade(double proposedTariff, double minimumTariff)
```

II) The Tariff class has the following attributes: destinationCountry (String), originCountry (String), productCategory (String), minimumTariff (double). It is assumed that String entries are recorded as a single word (_ is used to combine multiple words).

You are required to write the implementation of the Show class. Besides the usual mutator & accessor methods (*i.e.* getOrginCountry(), setMinimumTarrif(double)) the class must also have the following:

a) Parameterized constructor accepts four values and initializes destinationCountry, originCountry, productCategory, and minimumTariff to these passed values.

b) Copy constructor, which takes in only one parameter, a **Tariff** object. The newly created object will be assigned all the attributes of the passed object.
c) clone() method. This method will create and return a clone of the calling object.
d) toString() and an equals() methods. Two **Tariff**s are equal if they have the same attributes.

III) The **TariffList** class must implement the **TariffPolicy** interface, and it has the following:

(a) An inner class named **TariffNode**. This class has the following:
   i. Two private attributes: an object of **Tariff** and a pointer to a **TariffNode** object.
   ii. A default constructor, which assigns both attributes to null.
   iii. A parameterized constructor that accepts two parameters, a **Tariff** object and a **TariffNode** object, then initializes the attributes accordingly.
   iv. A copy constructor that creates deep copy of a **TariffNode**.
   v. A clone() method that creates deep copy of the node.
   vi. An equals() method that compares passed **TariffNode** with the current one.
   vii. Other mutator and accessor methods.

(b) A private attribute called head, which should point to the first **TariffNode** in this list.

(c) A private attribute called size, which always indicates the current size of the list.

(d) A default constructor, which creates an empty list.

(e) A copy constructor, which accepts a **TariffList** object and creates a copy of it.

(f) A method called addToStart(), which accepts one parameter, a **Tariff** object and then creates a **TariffNode** with that passed object and inserts it at the head of the list.

(g) A method called insertAtIndex(), which accepts two parameters, a **Tariff** object, and an integer representing an index. If the index is invalid (a valid index is between 0 and size-1), the method must throw a ***NoSuchElementException*** and terminate the program. If the index is valid, then the method creates a **TariffNode** with the passed **Tariff** object and inserts it at the given index.

(h) A method called deleteFromIndex(), which accepts one integer parameter representing an index. Again, if the index is invalid, the method must throw ***NoSuchElementException*** and terminate the program. Otherwise, the **TariffNode** pointed to by that index is deleted from the list.

(i) A method called deleteFromStart(), which deletes first **TariffNode** in the list (*i.e.* one pointed by the head). All special cases must be properly handled.

(j) A method called replaceAtIndex(), which accepts two parameters, a **Tariff** object and an integer representing an index. If the index is invalid, the method simply returns; otherwise, the object in the **TariffNode** at the passed index is to be replaced by the node created from the passed object.

(k) A method called find(), which accepts three parameters String origin, String destination and String category. The method then searches the list for a **TariffNode** with that **Tariff**. If such an object is found, then the method returns a pointer to that **TariffNode**; otherwise, method returns null. The method must keep track of how many iterations were made before the search finally finds **TariffNode** or concludes that it is not in the list.

(l) A method called contains(), which accepts three parameters String origin, String destination and String category. It returns true if a **TariffNode** with matching info is in the list; otherwise, the method returns false.

(m) A method called equals(), which accepts one parameter of type **TariffList**. The method returns true if the two lists contain similar **TariffNodes**; otherwise, the method returns false.

A sample **TariffList** is demonstrated in Figure 4.

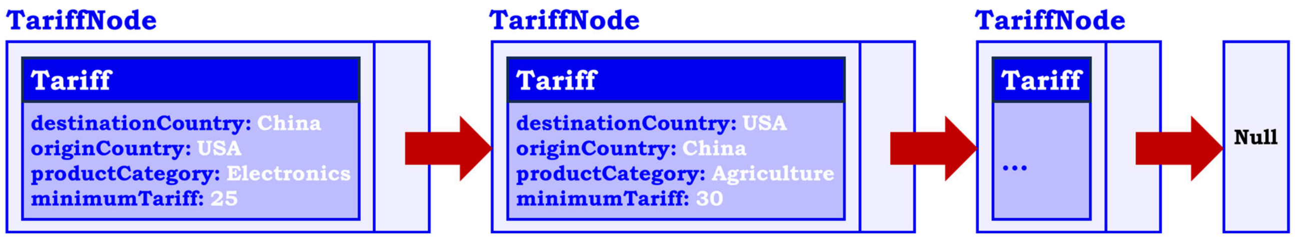


Figure 4. A sample TariffList (LinkedList implementation)

Finally, here are some general rules that you must consider when implementing the above methods:

-Whenever a node is added or deleted, the list size must be adjusted accordingly.

-All special cases must be handled, whether the method description explicitly states that or not.

-All clone () and copy constructors must perform a deep copy; no shallow copies are allowed.

-If any of your methods allows a privacy leak, you must clearly place a comment at the beginning of the method 1) indicating, this method may result in a privacy leak 2) explaining, reason behind the privacy leak. Please keep in mind that you are not required to implement these proposals.

IV) Now, you are required to write a public class named **TradeManager**. In the main() method, you must do the following:

(a) Create at least two empty lists from the **TariffList** class (needed for copy constructor III (e)).

(b) Open the Tariff.txt file and read its contents line by line. Use these records to initialize one of the **TariffList** objects you created above. You can use the addToStart () method to insert the read objects into the list. However, the ***list must not have any duplicate records***, hence, if the input file has duplicate entries, your code must handle this case so that each record is inserted in the list only once.

(c) Open TradeRequests.txt and create an Arraylist from the contents, then iterate through each of the **TariffRequest**s, process it and print the outcome whether it will be accepted, rejected or conditionally accepted. An output for the file in Figure 3 (b) is depicted below. Your program may assume the same file names but will be tested against other files.

(d) Prompt the user to enter a few **Tariff**s and search the list that you created from the input file for these values. Make sure to display the number of iterations performed.

(e) Following that, you must create enough objects to test each of the constructors/methods of your classes. The details of this part are left open to you. You can do whatever you wish to showcase all the methods created in this assignment are being tested, including some of the special cases.

```
REQ001 – Conditionally Accepted.
Proposed tariff 20% is within 20% of the required minimum tariff 25%.
A surcharge of $5000 is applied.

REQ002 – Accepted.
Proposed tariff meets or exceeds the minimum requirement.

REQ003 – Rejected
Proposed tariff 5% is more than 20% below the required minimum tariff 10%.
```

Figure 5. A sample outcome of the TradeRequests from Figure 3(b)

Some general information:

a. <u>You should open and close the Tariff.txt file only once; a better mark will be given for that</u>;
b. Do not use any external libraries or existing software to produce what is needed; that will directly result in a 0 mark!
c. Again, your program must work for any input files. The files provided with this assignment are only a possible version, and must not be considered as the general case when writing code.

You are allowed to work individually or in a group of a maximum of 2 students (*i.e.* you and one other student). Groups of more than 2 students will lead to zero (0) mark for each group member! You can work with a student from any section.

Submit only ONE version of an assignment. If more than one version is submitted; then the first one will be graded, and all others will be disregarded.

- Naming convention for zip file: Create one zip file, containing, Payroll, error and report files, Test cases, Java class files and produced documentation for your assignment, using the following naming convention:
- The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName_StudentID* is your name and ID number, respectively. Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. For example: for the first assignment, student Mike Simon, with ID 12345678, would submit a zip file named like: *a1_Mike-SIMON_12345678.zip.*
- If working in a group, only one of the group members can upload the assignment. Do **NOT** upload the file for each member! Add all IDs and names to the name of the file and inside each file in the zip file.

You **must submit** your assignment using **Moodle** ***under the Assignment 4 Submission folder***. Assignments uploaded to an incorrect submission folder will not be checked and will result in a mark of 0 as if no submission was made.

<u>**IMPORTANT**</u>: make sure you submit way before the closing date to avoid "surprises". Please also note that we cannot submit it for you as the submission must come from your account. We will not accept any submission by email for whatever reason. Excuses like "my system crashed when I was submitting and when it restarted it was past the deadline by 2 minutes", "I thought my friend will submit but they did not", "I forgot to submit", "I suddenly slept and woke up 3 minutes after the closing time", or similar will not be accepted and will go <u>**unanswered**</u>.

IMPORTANT (Please read very carefully): Additionally, which is very important, a demo will take place with the markers afterwards. Markers will inform you about the details of demo time and how to book a time slot for your demo. ***If working in a group, both members must be present during demo*** time. ***Different marks may be assigned to teammates based on this demo***.

If you ***fail to demo***, a ***zero mark is assigned*** regardless of your submission.

If you book a demo time, and ***do not show up***, for whatever reason, you will be allowed to reschedule a second demo but a **penalty of 50%** will be applied.

Failing to demo at the second appointment will result in **zero marks** and ***no more chances*** will be given under any conditions.

Submitting Assignment 3

Evaluation Criteria for Assignment 4 (10 points)

Total	**10 pts**
Part 1	**3 pts**
Correct Implementation	2 pts
Proper Use & Testing of the Code	1 pt
Part 2	**7 pts**
Task I	1 pt
Task II	2 pts
Task III	2 pts
Task IV	2 pts