

TRABAJO COLABORATIVO – ENTREGA 2

INSTITUCIÓN UNIVERSITARIA POLITÉCNICO GRANCOLOMBIANO
CONCEPTOS FUNDAMENTALES DE PROGRAMACION

TUTOR

DIEGO HERNAN ROA ORDOÑEZ

INTEGRANTES

DIEGO FERNANDO REINA RAMIREZ – 100227163

ROSA OSPINO LOPEZ – COD 1065827855

ANDREA AGUDELO MEJIA – COD 1000282440

LINA MARIA PINZON FARFAN – COD 100232376

LIDIA DAMARIS GARZON MORERA – COD 100252365

ENTREGA 1

En la primera entrega se espera que el repositorio compartido tenga las clases y métodos necesarios en un proyecto de Eclipse, para generar archivos planos como los que recibiría como entrada el problema principal del proyecto. Es decir, esta entrega consiste en diseñar e implementar un programa que al correrse genere una serie de archivos que sirvan como entrada al programa que organizará los datos. La primera entrega es, entonces, básicamente el diseño e implementación de la clase GenerateInfoFiles.

Para el desarrollo de esta parte del programa, se tienen 3 archivos de entrada, se implementan los métodos de lectura y creación de archivos de salida, los cuales se crean dentro de una carpeta con nombre outputFiles

También se realiza la implementación de los métodos propuestos en la guía

- createSalesMenFile(int randomSalesCount, String name, long id): dada una cantidad, un nombre y un id, crea un archivo pseudoaleatorio de ventas de un vendedor con el nombre y el id dados.
- createProductsFile(int productsCount): crea un archivo con información pseudoaleatoria de productos, con los datos de productsCount productos.
- createSalesManInfoFile(int salesmanCount): crea un archivo con información de salesmanCount vendedores; el número de estos según lo indique el argumento entero. La información debe ser generada de manera pseudoaleatoria y ser coherente, es decir, los nombres y apellidos pueden ser extraídos de listas de nombres reales de personas.

Enlace del repositorio en GitHub:

https://github.com/HardSoft2021/Poli-flat_file-generation.git

The screenshot shows an IDE window titled "GenerateInfoFiles.java X". The editor displays the following code:

```
2  /*
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
4   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
5   */
6  package generateinfofiles;
7
8  /**
9   *
10   * @author diego reina, rosa ospino, andrea agudelo
11   */
12  import java.io.BufferedReader;
13  import java.io.FileReader;
14  import java.io.FileWriter;
15  import java.io.IOException;
16  import java.nio.file.Files;
17  import java.nio.file.Path;
18  import java.nio.file.Paths;
19  import java.util.ArrayList;
20  import java.util.List;
21  import java.util.Random;
22
23  /**
24   * This class provides methods for generating seller, product, and sales
25   * information files.
```

Below the code, the "Output - GenerateInfoFiles (run)" window shows the following message:

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

The screenshot shows the same IDE window, but with the following code visible:

```
33  public static void main(String[] args) {
34      GenerateInfoFiles generator = new GenerateInfoFiles();
35      try {
36          // Generate information files for sellers, products and sales
37          generator.createSalesManInfoFile(5, "sellers.txt");
38          generator.createProductsFile(10, "products.txt");
39          generator.createSalesMenFile(10, "sales.txt", "sellers.txt");
40      } catch (IOException e) {
41          System.err.println("Error generating files: " + e.getMessage());
42      }
43  }
44
45  /**
46   *
47   * Output directory to store the generated files.
48   */
49  private static final String OUTPUT_FOLDER = "outputFiles/";//A directory is created as a relative path
50
51  /**
52   *
53   * Create a seller information file with pseudo-random data.
54   *
55   * @param salesmanCount Number of sellers to generate.
56   * @param sellersFile Name of the file that contains information sellers.
57   * @throws IOException If an error occurs while writing the file.
58   */
59  public void createSalesManInfoFile(int salesmanCount, String sellersFile) throws IOException {
60      // Read seller information from the file
61      List<String> sellersInfo = readLinesFromFile(sellersFile);
62      if (sellersInfo.isEmpty()) {
```

```
GenerateInfoFiles.java x
Source History
62     if (sellersInfo.isEmpty()) {
63         throw new IOException("Seller file is empty");
64     }
65
66     // Generate salespeople pseudo-randomly and write to the output file
67     Random random = new Random();
68     createFolderIfNotExists(OUTPUT_FOLDER);
69     try (FileWriter writer = new FileWriter(OUTPUT_FOLDER + "salesman_info.txt")) {
70         for (int i = 0; i < salesmanCount; i++) {
71             String seller = sellersInfo.get(random.nextInt(sellersInfo.size()));
72             writer.write(seller + "\n");
73         }
74     }
75 }
76
77 /**
78  * Create a product information file with pseudo-random data.
79  *
80  * @param productsCount Number of products to generate.
81  * @param productsFile Name of the file containing product information
82  * products.
83  * @throws IOException If an error occurs while writing the file.
84  */
85 public void createProductsFile(int productsCount, String productsFile) throws IOException {
86     // Read product information from file
87     List<String> productsInfo = readLinesFromFile(productsFile);
88     if (productsInfo.isEmpty()) {
89         throw new IOException("Product file is empty");
90     }
91 }
```

```
GenerateInfoFiles.java x
Source History
92 // Generate products pseudorandomly and write to the output file
93 Random random = new Random();
94 createFolderIfNotExists(OUTPUT_FOLDER);
95 try (FileWriter writer = new FileWriter(OUTPUT_FOLDER + "products_info.txt")) {
96     for (int i = 0; i < productsCount; i++) {
97         String product = productsInfo.get(random.nextInt(productsInfo.size()));
98         writer.write(product + "\n");
99     }
100 }
101
102 /**
103  * Create a sales information file with pseudo-random data.
104  *
105  * @param randomSalesCount Number of random sales to generate.
106  * @param salesFile Name of the file containing sales information.
107  * @param sellersFile Name of the file that contains sales information
108  * sellers.
109  * @throws IOException If an error occurs while writing the file.
110  */
111
112 public void createSalesMenFile(int randomSalesCount, String salesFile, String sellersFile) throws IOException {
113     // Read seller information from the file
114     List<String> sellersInfo = readLinesFromFile(sellersFile);
115     if (sellersInfo.isEmpty()) {
116         throw new IOException("Seller file is empty");
117     }
118
119     // Generate random sales and write to output file
120     Random random = new Random();
121     createFolderIfNotExists(OUTPUT_FOLDER);
122 }
```

```
GenerateInfoFiles.java X
Source History
120 Random random = new Random();
121 createFolderIfNotExists(OUTPUT_FOLDER);
122 try (BufferedReader reader = new BufferedReader(new FileReader(salesFile)); FileWriter writer = new FileWriter(OUTPUT_FOLDER + "sales_info.txt")) {
123
124     String line;
125     for (int i = 0; i < randomSalesCount; i++) {
126         String seller = sellersInfo.get(random.nextInt(sellersInfo.size()));
127         writer.write(seller + "\n");
128
129         // Read sales from sales file
130         List<String> sales = readLinesFromFile(salesFile);
131         for (String sale : sales) {
132             writer.write(sale + "\n");
133         }
134     }
135 }
136
137
138 /**
139  * Reads all lines from a file and returns them as a list of chains.
140  *
141  * @param filename Name of the file to read.
142  * @return List of lines read from the file.
143  * @throws IOException If an error occurs while reading the file.
144  */
145 private List<String> readLinesFromFile(String filename) throws IOException {
146     List<String> lines = new ArrayList<>();
147     try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
148         String line;
149         while ((line = reader.readLine()) != null) {
```

```
GenerateInfoFiles.java X
Source History
120 Random random = new Random();
121 createFolderIfNotExists(OUTPUT_FOLDER);
122 try (BufferedReader reader = new BufferedReader(new FileReader(salesFile)); FileWriter writer = new FileWriter(OUTPUT_FOLDER + "sales_info.txt")) {
123
124     String line;
125     for (int i = 0; i < randomSalesCount; i++) {
126         String seller = sellersInfo.get(random.nextInt(sellersInfo.size()));
127         writer.write(seller + "\n");
128
129         // Read sales from sales file
130         List<String> sales = readLinesFromFile(salesFile);
131         for (String sale : sales) {
132             writer.write(sale + "\n");
133         }
134     }
135 }
136
137
138 /**
139  * Reads all lines from a file and returns them as a list of chains.
140  *
141  * @param filename Name of the file to read.
142  * @return List of lines read from the file.
143  * @throws IOException If an error occurs while reading the file.
144  */
145 private List<String> readLinesFromFile(String filename) throws IOException {
146     List<String> lines = new ArrayList<>();
147     try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
148         String line;
149         while ((line = reader.readLine()) != null) {
```

A continuación se evidencia la carpeta de salida outputFiles donde se guarda la generación de archivos, teniendo en cuenta los 3 archivos planos (producto.txt, sales.txt, Sellers.txt) como método de ingreso de información.

Nombre	Fecha de modificación	Tipo	Tamaño
build	23/03/2024 8:55 p. m.	Carpeta de archivos	
nbproject	23/03/2024 8:53 p. m.	Carpeta de archivos	
outputFiles	24/03/2024 9:37 p. m.	Carpeta de archivos	
src	23/03/2024 8:53 p. m.	Carpeta de archivos	
test	24/03/2024 5:40 p. m.	Carpeta de archivos	
build.xml	23/03/2024 8:53 p. m.	Microsoft Edge H...	4 KB
manifest.mf	23/03/2024 8:53 p. m.	Archivo MF	1 KB
products.txt	24/03/2024 6:28 p. m.	Documento de te...	1 KB
sales.txt	24/03/2024 6:06 p. m.	Documento de te...	1 KB
sellers.txt	24/03/2024 9:44 p. m.	Documento de te...	1 KB

products_info.txt	24/03/2024 9:37 p. m.	Documento de te...	1 KB
sales_info.txt	24/03/2024 9:37 p. m.	Documento de te...	1 KB
salesman_info.txt	24/03/2024 9:37 p. m.	Documento de te...	1 KB

```
*salesman_info.txt: Bl...
Archivo Edición Formato Ver Ayuda
CC;987654321;Rosa;Ospino
CC;987654321;Andrea;Agudelo
CC;123412348;Juana;DeArco
CC;123412348;Juana;DeArco
CC;11449245;Diego;Reina
100% UNIX (LF) UTF-8
```


```
product...
Archivo Edición Formato Ver
Ayuda
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
2;mouse;25.000
1;keyboard_usb;30.000
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
2;mouse;25.000
2;mouse;25.000
UNIX (LF) UTF-8
```


```
products_info.t...
Archivo Edición Formato Ver Ayuda
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
5;ipad;1.500.000
2;mouse;25.000
1;keyboard_usb;30.000
5;ipad;1.500.000
5;ipad;1.500.000
2;mouse;25.000
2;mouse;25.000
10 UNIX (LF) UTF-8
```


ENTREGA 2

Esta entrega consiste en una versión preliminar del proyecto completo, con posibles elementos faltantes que pueden depender de las dinámicas de evolución de la solución en cada grupo de trabajo. Debe tener un documento adjunto indicando qué partes le faltan al proyecto.

Para esta parte del desarrollo se logró hacer la implementación de los archivos de salida en formato csv

 products_info.csv

 sales_info.csv

 salesman_info.csv

Los cuales al ser ejecutados con los siguientes métodos

- a. createSalesMenFile(int randomSalesCount, String name, long id): dada una cantidad, un nombre y un id, crea un archivo pseudoaleatorio de ventas de un vendedor con el nombre y el id dados.
- b. createProductsFile(int productsCount): crea un archivo con información pseudoaleatoria de productos, con los datos de productsCount productos.
- c. createSalesManInfoFile(int salesmanCount): crea un archivo con información de salesmanCount vendedores; el número de estos según lo indique el argumento entero. La información debe ser generada de manera pseudoaleatoria y ser coherente, es decir, los nombres y apellidos pueden ser extraídos de listas de nombres reales de personas

Puntos pendientes para la próxima entrega

- El programa debe crear un archivo con la información de todos los vendedores, de a uno por línea. Al frente del nombre de cada vendedor, separado por punto y coma, debe estar la cantidad de dinero que recaudó según los archivos. El archivo debe estar ordenado por cantidad de dinero, de mayor a menor, de a un vendedor por línea. Es básicamente un archivo de reporte de ventas de los vendedores, del mejor al peor; un archivo CSV.
- El programa debe crear un archivo con la información de los productos vendidos por cantidad, ordenados en forma descendente. Deben ir el nombre y el precio, separados por punto y coma, y de a un producto por línea. Es básicamente un archivo plano CSV

- a. La posibilidad de procesar más de un archivo por vendedor.
- b. La posibilidad de trabajar con archivos serializados.
- c. La posibilidad de detectar archivos con formato erróneo o con información incoherente, como un id de producto que no exista, o precios o cantidades negativas

Codigo fuente clase GenerateInfoFiles

```
package generateinfofiles;

import java.io.*;
import java.nio.file.*;
import java.util.*;

/**
 * This class provides methods for generating seller, product, and sales information files.
 */
public class GenerateInfoFiles {

    /** Output directory to store the generated files. */
    private static final String OUTPUT_FOLDER = "outputFiles/";

    /**
     * Main method to run the information file generator.
     * @param args Command line arguments (not used).
     */
    public static void main(String[] args) {
        GenerateInfoFiles generator = new GenerateInfoFiles();
        try {
            // Generate information files for sellers, products and sales
            generator.createSalesManInfoFile(5, "sellers.txt");
            generator.createProductsFile(10, "products.txt");
        }
    }
}
```



```

        generator.createSalesMenFile(10, "sales.txt", "sellers.txt");
    } catch (IOException e) {
        System.err.println("Error generating files: " + e.getMessage());
    }
}

/**
 * Create a seller information file with pseudo-random data.
 * @param salesmanCount Number of sellers to generate.
 * @param sellersFile Name of the file that contains information sellers.
 * @throws IOException If an error occurs while writing the file.
 */
public void createSalesManInfoFile(int salesmanCount, String sellersFile) throws IOException {
    // Read seller information from the file
    List<String> sellersInfo = readLinesFromFile(sellersFile);
    if (sellersInfo.isEmpty()) {
        throw new IOException("Seller file is empty");
    }

    // Generate salespeople pseudo-randomly and write to the output file
    Random random = new Random();
    createFolderIfNotExists(OUTPUT_FOLDER);
    try (FileWriter writer = new FileWriter(OUTPUT_FOLDER + "salesman_info.csv")) {
        for (int i = 0; i < salesmanCount; i++) {
            String seller = sellersInfo.get(random.nextInt(sellersInfo.size()));
            // Simulate sales amount (random)
            double salesAmount = random.nextDouble() * 10000;
            writer.write(seller + ";" + String.format("%.2f", salesAmount) + "\n");
        }
    }
}

```

```

    }
}

/**
 * Create a product information file with pseudo-random data.
 * @param productsCount Number of products to generate.
 * @param productsFile Name of the file containing product information products.
 * @throws IOException If an error occurs while writing the file.
 */
public void createProductsFile(int productsCount, String productsFile) throws IOException {
    // Read product information from file
    List<String> productsInfo = readLinesFromFile(productsFile);
    if (productsInfo.isEmpty()) {
        throw new IOException("Product file is empty");
    }

    // Generate products pseudorandomly and write to the output file
    Random random = new Random();
    createFolderIfNotExists(OUTPUT_FOLDER);
    try (FileWriter writer = new FileWriter(OUTPUT_FOLDER + "products_info.csv")) {
        for (int i = 0; i < productsCount; i++) {
            String product = productsInfo.get(random.nextInt(productsInfo.size()));
            // Simulate product price (random)
            double price = random.nextDouble() * 100;
            writer.write(product + ";" + String.format("%.2f", price) + "\n");
        }
    }
}

```

```

/**
 * Create a sales information file with pseudo-random data.
 * @param randomSalesCount Number of random sales to generate.
 * @param salesFile Name of the file containing sales information.
 * @param sellersFile Name of the file that contains sales information sellers.
 * @throws IOException If an error occurs while writing the file.
 */

public void createSalesMenFile(int randomSalesCount, String salesFile, String sellersFile) throws
IOException {

    // Read seller information from the file
    List<String> sellersInfo = readLinesFromFile(sellersFile);

    if (sellersInfo.isEmpty()) {
        throw new IOException("Seller file is empty");
    }

    // Generate random sales and write to output file
    Random random = new Random();
    createFolderIfNotExists(OUTPUT_FOLDER);
    try (FileWriter writer = new FileWriter(OUTPUT_FOLDER + "sales_info.csv")) {

        for (int i = 0; i < randomSalesCount; i++) {
            String seller = sellersInfo.get(random.nextInt(sellersInfo.size()));
            // Simulate sales data (random)
            String saleData = generateRandomSalesData();
            writer.write(seller + ";" + saleData + "\n");
        }
    }
}

```

```

/**
 * Generates random sales data in the format: productName;quantity;price.
 * @return Randomly generated sales data.
 */
private String generateRandomSalesData() {
    Random random = new Random();

    // Let's assume we have a list of product names
    List<String> productNames = Arrays.asList("keyboard_usb", "mouse", "notebook", "memory",
"ipad");

    String productName = productNames.get(random.nextInt(productNames.size()));
    int quantity = random.nextInt(10) + 1; // Random quantity between 1 and 10
    double price = random.nextDouble() * 100; // Random price between 0 and 100
    return productName + ";" + quantity + ";" + String.format("%.2f", price);
}

```

```

/**
 * Reads all lines from a file and returns them as a list of strings.
 * @param filename Name of the file to read.
 * @return List of lines read from the file.
 * @throws IOException If an error occurs while reading the file.
 */
private List<String> readLinesFromFile(String filename) throws IOException {
    List<String> lines = new ArrayList<>();

    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        String line;
        while ((line = reader.readLine()) != null) {
            lines.add(line);
        }
    }
}

```

```
        return lines;
    }

    /**
     * Creates a folder if it does not exist in the specified path.
     * @param folderPath Path of the folder to create.
     * @throws IOException If an error occurs while creating the folder.
     */
    private void createFolderIfNotExists(String folderPath) throws IOException {
        Path path = Paths.get(folderPath);
        if (!Files.exists(path)) {
            Files.createDirectories(path);
        }
    }
}
```

Fin del documento