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

```
Generatinfofficiaya x

Click Indias/Inbhost/SystemFileSystem/Templates/Licenses/Hicenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Licenses/Lice
```

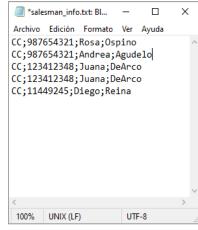
```
| Source | Hotoy | Property | Pro
```

```
Source Hotoy Brain Pew Random () creater of the state of
```

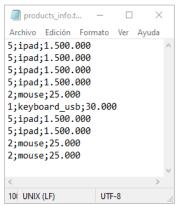
A continuación se evidencia la carpeta de salida outputFiles donde se guarda la generación de archivos, teniendo en cuenta los 3 archivos planos (producto.txt, sales.txt, Sellers.txt) como método de ingreso de información.

| Nombre | Fecha de modificación | Тіро | Tamaño |
|--------------|-----------------------|---------------------|--------|
| uild 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 | |
| e 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 |





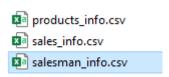




ENTREGA 2

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



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