# Assignment 5:

### (total 70 points)

### Exercise 1 File I/O(2points)

|  |  |
| --- | --- |
|  | Using File I/OPrerequisites, Goals, and Outcomes ***Prerequisites:*** Before you begin this exercise, you need mastery of the following:   * Java API   + Knowledge of the class StringTokenizer * File I/O   + Knowledge of file I/O     - How to read data from a file     - How to write data to a file   ***Goals:*** Reinforce your ability to use file I/O  ***Outcomes:*** You will master the following skills:   * Produce applications that read data from a file and parse it * Produce applications that write data to a file  Background This assignment asks you to implement two methods: one that reads employee data from a file and another that writes employee data to a file. The employee data contains basic information (ID, name, and salary) for a collection of employees. Description In this assignment, you will finish the implementation of EmployeeFileIO. iCarnegie will provide a test driver and the class Employee. Class Employee A complete implementation of this class is included in the student archive [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/student-files.zip). Stop now and review its documentation:   * [Employee.html](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/Employee.html). Documentation for class Employee  Class EmployeeFileIO A partial implementation of this class is included in the student archive [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/student-files.zip). You should complete the implementation of the following methods:   * public static ArrayList<Employee> read(String fileName) * throws FileNotFoundException, * IOException, * NoSuchElementException, * NumberFormatException   This method creates an ArrayList of Employee objects from a file that contains employee data.   * public static void write(String fileName, ArrayList<Employee> arrayList) * throws IOException   This method creates a file of employee data from an ArrayList of Employee objects. Class TestEmployeeFileIO This class is a test driver for EmployeeFileIO. It contains test cases for each method in EmployeeFileIO. A complete implementation is included in the student archive [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/student-files.zip). You should use this class to test your implementation of EmployeeFileIO.  Your implementation of method read is tested by comparing the ArrayList returned by your implementation against an ArrayList returned by the iCarnegie implementation. In the same way, your implementation of method write is tested comparing the file produced by your implementation against a file produced by the iCarnegie implementation. Class TestHelper This class contains auxiliary methods used by the test driver: a method for comparing two ArrayList objects and a method for comparing two files. A complete and compiled implementation is included in the student archive [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/student-files.zip).  Review its documentation and become familiar with it:   * [TestHelper.html](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/TestHelper.html). Documentation for class TestHelper  Files The following files are needed to complete this assignment:   * [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-qz-pr-file-io/pool-pr-file-io/qn-pr-io-empl/handout/student-files.zip) — Download this file. This archive contains the following:   + Class files     - TestHelper.class   + Documentation     - *Employee.html*     - TestHelper.html   + Java files     - *Employee.java*. A complete implementation     - *TestEmployeeFileIO.java*. A complete implementation     - *EmployeeFileIO.java*. Use this template to complete your implementation.   + Data files used by the test driver     - empty.txt. An empty file     - employees.txt. A file with employee data  Tasks Implement the methods read and write in class EmployeeFileIO. Document your code using Javadoc and follow Sun's code conventions. The following steps will guide you through this assignment. Work incrementally and test each increment. Save often.   1. **Extract** the files from student-files.zip 2. **Test** each method as soon as you finish writing it by issuing the following command at the command prompt:   C:\>java TestEmployeeFileIO   1. **Implement** the method read**(10 points)**: It begins by creating an empty ArrayList and a [BufferedReader](http://java.sun.com/j2se/1.5/docs/api/java/io/BufferedReader.html) object to read data from the specified file. It then proceeds to read each line in the file. After it reads a line, it extracts the ID, name, and salary of an employee, creates an Employee object, and adds the new object to the end of the ArrayList. When all data has been read, it returns the ArrayList. Use [BufferedReader.readLine](http://java.sun.com/j2se/1.5/docs/api/java/io/BufferedReader.html#readLine()) to read the data in the file.  Use [java.util.StringTokenizer](http://java.sun.com/j2se/1.5/docs/api/java/util/StringTokenizer.html) to parse the data.   You can assume that every line in the file contains the data for exactly one employee in the following format:  ID\_name\_salary  where:   * + ID is an integer that represents the ID of the employee.   + name is a String that represents the name of the employee.   + salary is a double that represents the salary of the employee.   The fields are delimited by an underscore ( \_ ). You can assume that the fields themselves do not contain any underscores.  The method read should not contain try-catch blocks for the following exceptions. This requirement will simplify the code.   * FileNotFoundException. Thrown by the BufferedReader constructor if the specified file does not exist. * IOException. Thrown by BufferedReader.readLine if an I/O error occurs. * NoSuchElementException. Thrown by StringTokenizer.nextToken if the specified file contains incomplete data. * NumberFormatException. Thrown by Integer.parseInt and Double.parseDouble if the specified file contains invalid data.  1. **Implement** the method write**(10 points)**: It first creates a [PrintWriter](http://java.sun.com/j2se/1.5/docs/api/java/io/PrintWriter.html) object for writing data to the specified file (if the file does not exist, one will be created). It then writes the ID, name, and salary of each employee in the specified ArrayList to the specified file. Every line in the file should contain the data for exactly one employee in the following format:   ID\_name\_salary  where:   * + ID is an integer that represents the ID of the employee.   + name is a String that represents the name of the employee.   + salary is a double that represents the salary of the employee.   The fields are delimited by an underscore ( \_ ).  The order of the employees in the file should match the order of the employees in the ArrayList. If the specified file exists, its contents should be erased when it is opened for writing.  The method write should not contain a try-catch block for the following exception. This requirement will simplify the code.   * IOException. Thrown by the FileWriter constructor if the specified file could not be found or created.  Submission Upon completion, submit **only** the following.   1. EmployeeFileIO.java, EmployeeFileIO.class 2. a word file including the program running results   inherit |

### Exercise 2 File I/O in the Gourmet Coffee (30 points)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Using File I/O in the Gourmet Coffee SystemPrerequisites, Goals, and Outcomes ***Prerequisites:*** Before you begin this exercise, you need mastery of the following:   * Java API   + Knowledge of the class StringTokenizer * File I/O   + Knowledge of file I/O     - How to read data from a file     - How to write data to a file   ***Goals:*** Reinforce your ability to use file I/O  ***Outcomes:*** You will master the following skills:   * Produce applications that read data from a file and parse it * Produce applications that write data to a file  Background In this assignment, you will create another version of the Gourmet Coffee System. In previous versions, the data for the product catalog was hard-coded in the application. In this version, the data will be loaded from a file. Also, the user will be able to write the sales information to a file in one of three formats: plain text, HTML, or XML. Part of the work has been done for you and is provided in the student archive. You will implement the code that loads the product catalog and persists the sales information. Description The Gourmet Coffee System sells three types of products: coffee, coffee brewers, and accessories for coffee consumption. A file called catalog.dat stores the product data:   * [catalog.dat](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-exer-file-io/pool-pr-io-prgrmng/qn-pr-io-gou-cof/handout/catalog.dat). File with product data   Every line in catalog.dat contains exactly one product.  A line for a coffee product has the following format:  Coffee\_code\_description\_price\_origin\_roast\_flavor\_aroma\_acidity\_body  where:   * "Coffee" is a prefix that indicates the line type. * code is a string that represents the code of the coffee. * description is a string that represents the description of the coffee. * price is a double that represents the price of the coffee. * origin is a string that represents the origin of the coffee. * roast is a string that represents the roast of the coffee. * flavor is a string that represents the flavor of the coffee. * aroma is a string that represents the aroma of the coffee. * acidity is a string that represents the acidity of the coffee. * body is a string that represents the body of the coffee.   The fields are delimited by an underscore ( \_ ). You can assume that the fields themselves do not contain any underscores.  A line for a coffee brewer has the following format:  Brewer\_code\_description\_price\_model\_waterSupply\_numberOfCups  where:   * "Brewer" is a prefix that indicates the line type. * code is a string that represents the code of the brewer. * description is a string that represents the description of the brewer. * price is a double that represents the price of the brewer. * model is a string that represents the model of the coffee brewer. * waterSupply is a string that represents the water supply of the coffee brewer. * numberOfCups is an integer that represents the capacity of the coffee brewer in number of cups.   The fields are delimited by an underscore ( \_ ). You can assume that the fields themselves do not contain any underscores.  A line for a coffee accessory has the following format:  Product\_code\_description\_price  where:   * "Product" is a prefix that indicates the line type. * code is a string that represents the code of the product. * description is a string that represents the description of the product. * price is a double that represents the price of the product.   The fields are delimited by an underscore ( \_ ). You can assume that the fields themselves do not contain any underscores.  The following class diagram highlights the elements you will use to load the product catalog and persist the sales information:   |  | | --- | | Figure 1 Portion of Gourmet Coffee System class diagram | | **Figure 1** Portion of Gourmet Coffee System class diagram |   In this assignment, you will implement FileCatalogloader and complete the implementation of GourmetCoffee. Interface CatalogLoader The interface CatalogLoader declares a method for producing a product catalog. A complete implementation of this interface is provided in the student archive.  Method:   * Catalog loadCatalog(String fileName) * throws FileNotFoundException, * IOException, * DataFormatException   Loads the information in the specified file into a product catalog and returns the catalog. Class DataFormatException This exception is thrown when a line in the file being parsed has errors:   * The line does not have the expected number of tokens. * The tokens that should contain numbers do not.   A complete implementation of this class is provided in the student archive. Class FileCatalogLoader The class FileCatalogLoader implements interface CatalogLoader. It is used to obtain a product catalog from a file. You should implement this class from scratch:  Methods:   * private Product readProduct(String line) * throws DataFormatException   This method reads a line of coffee-accessory data. It uses the class StringTokenizer to extract the accessory data in the specified line. If the line is error free, this method returns a Product object that encapsulates the accessory data. If the line has errors, that is, if it does not have the expected number of tokens or the token that should contain a double does not; this method throws a DataFormatException that contains the line of malformed data.   * private Coffee readCoffee(String line)) * throws DataFormatException   This method reads a line of coffee data. It uses the class StringTokenizer to extract the coffee data in the specified line. If the line is error free, this method returns a Coffee object that encapsulates the coffee data. If the line has errors, that is, if it does not have the expected number of tokens or the token that should contain a double does not; this method throws a DataFormatException that contains the line of malformed data.   * private CoffeeBrewer readCoffeeBrewer(String line) * throws DataFormatException   This method reads a line of coffee-brewer data. It uses the class StringTokenizer to extract the brewer data in the specified line. If the line is error free, this method returns a CoffeeBrewer object that encapsulates the brewer data. If the line has errors, that is, if it does not have the expected number of tokens or the tokens that should contain a number do not; this method throws a DataFormatException that contains the line of malformed data.   1. public Catalog loadCatalog(String filename) 2. throws FileNotFoundException, 3. IOException, 4. DataFormatException   This method loads the information in the specified file into a product catalog and returns the catalog. It begins by opening the file for reading. It then proceeds to read and process each line in the file. The method [String.startsWith](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html#startsWith(java.lang.String)) is used to determine the line type:   * + If the line type is "Product", the method readProduct is invoked.   + If the line type is "Coffee", the method readCoffee is invoked.   + If the line type is "Brewer", the method readCoffeeBrewer is invoked.   After the line is processed, loadCatalog adds the product (accessory, coffee, or brewer) to the product catalog. When all the lines in the file have been processed, load returns the product catalog to the calling method.  This method can throw the following exceptions:   * FileNotFoundException — if the specified file does not exist. * IOException — if there is an error reading the information in the specified file. * DataFormatException — if a line in the file has errors (the exception should contain the line of malformed data).  Class GourmetCoffee A partial implementation of class GourmetCoffee is provided in the student archive. You should implement writeFile, a method that writes sales information to a file:   * private void writeFile(String fileName, String content) * throws IOException   This method creates a new file with the specified name, writes the specified string to the file, and then closes the file. Class TestFileCatalogLoader This class is a test driver for FileCatalogLoader. A complete implementation is included in the student archive [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-exer-file-io/pool-pr-io-prgrmng/qn-pr-io-gou-cof/handout/student-files.zip). You should use this class to test your implementation of FileCatalogLoader. Files The following files are needed to complete this assignment:   * [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-io-prgrmng/assm-exer-file-io/pool-pr-io-prgrmng/qn-pr-io-gou-cof/handout/student-files.zip) — Download this file. This archive contains the following:   + Class files     - *Coffee.class*     - *CoffeeBrewer.class*     - *Product.class*     - *Catalog.class*     - *OrderItem.class*     - *Order.class*     - *Sales.class*     - *SalesFormatter.class*     - *PlainTextSalesFormatter.class*     - *HTMLSalesFormatter.class*     - *XMLSalesFormatter.class*   + Documentation     - *Coffee.html*     - *CoffeeBrewer.html*     - *Product.html*     - *Catalog.html*     - *OrderItem.html*     - *Order.html*     - *Sales.html*     - *SalesFormatter.html*     - *PlainTextSalesFormatter.html*     - *HTMLSalesFormatter.html*     - *XMLSalesFormatter.html*   + Java files     - *CatalogLoader.java*. A complete implementation     - *DataFormatException.java*. A complete implementation     - TestFileCatalogLoader.java. A complete implementation     - *GourmetCoffee.java*. Use this template to complete your implementation.   + Data files for the test driver     - *catalog.dat*. A file with product information     - *empty.dat*. An empty file  Tasks Implement the class FileCatalogLoader and the method GourmetCoffee.writeFile. Document using Javadoc and follow Sun's code conventions. The following steps will guide you through this assignment. Work incrementally and test each increment. Save often.   1. **Extract** the files from student-files.zip 2. **Then**, implement class FileCatalogLoader from scratch. Use the TestFileCatalogLoader driver to test your implementation. (20 points) 3. **Next**, implement the method GourmetCoffee.writeFile. (10 points) 4. **Finally**, compile the class GourmetCoffee, and execute the class GourmetCoffee by issuing the following command at the command prompt:   C:\>java GourmetCoffee catalog.dat  Sales information has been hard-coded in the GourmetCoffee template provided by iCarnegie.   * + If the user displays the catalog, the output should be:   C001 Colombia, Whole, 1 lb C002 Colombia, Ground, 1 lb C003 Italian Roast, Whole, 1 lb C004 Italian Roast, Ground, 1 lb C005 French Roast, Whole, 1 lb C006 French Roast, Ground, 1 lb C007 Guatemala, Whole, 1 lb C008 Guatemala, Ground, 1 lb C009 Sumatra, Whole, 1 lb C010 Sumatra, Ground, 1 lb C011 Decaf Blend, Whole, 1 lb C012 Decaf Blend, Ground, 1 lb B001 Home Coffee Brewer B002 Coffee Brewer, 2 Warmers B003 Coffee Brewer, 3 Warmers B004 Commercial Coffee, 20 Cups B005 Commercial Coffee, 40 Cups A001 Almond Flavored Syrup A002 Irish Creme Flavored Syrup A003 Mint Flavored syrup A004 Caramel Flavored Syrup A005 Gourmet Coffee Cookies A006 Gourmet Coffee Travel Thermo A007 Gourmet Coffee Ceramic Mug A008 Gourmet Coffee 12 Cup Filters A009 Gourmet Coffee 36 Cup Filters   * + If the user saves the sales information in plain text, a file with the following content should be created:   + ------------------------   + Order 1   + 5 C001 17.99   + Total = 89.94999999999999   + ------------------------   + Order 2   + 2 C002 18.75   + 2 A001 9.0   + Total = 55.5   + ------------------------   + Order 3   + 1 B002 200.0   Total = 200.0   * + If the user saves the sales information in HTML, a file with the following content should be created:   <html>  <body>  <center><h2>Orders</h2></center>  <hr>  <h4>Total = 89.94999999999999</h4>  <p>  <b>code:</b> C001<br>  <b>quantity:</b> 5<br>  <b>price:</b> 17.99  </p>  <hr>  <h4>Total = 55.5</h4>  <p>  <b>code:</b> C002<br>  <b>quantity:</b> 2<br>  <b>price:</b> 18.75  </p>  <p>  <b>code:</b> A001<br>  <b>quantity:</b> 2<br>  <b>price:</b> 9.0  </p>  <hr>  <h4>Total = 200.0</h4>  <p>  <b>code:</b> B002<br>  <b>quantity:</b> 1<br>  <b>price:</b> 200.0  </p>  </body> </html>   * + If the user saves the sales information in XML, a file with the following content should be created:   <Sales>  <Order total="89.94999999999999">  <OrderItem quantity="5" price="17.99">C001</OrderItem>  </Order>  <Order total="55.5">  <OrderItem quantity="2" price="18.75">C002</OrderItem>  <OrderItem quantity="2" price="9.0">A001</OrderItem>  </Order>  <Order total="200.0">  <OrderItem quantity="1" price="200.0">B002</OrderItem>  </Order> </Sales> Submission Upon completion, submit **only** the following:   1. FileCatalogLoader.java, FileCatalogLoader.class 2. GourmetCoffee.java , GourmetCoffee.class 3. a word file including the program running results |

### Exercise 3(Swing GUI of Gourmet Coffee System)(20 points)

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Implementing a GUI for the Gourmet Coffee System (Part I)Prerequisites, Goals, and Outcomes ***Prerequisites:*** Before you begin this exercise, you need mastery of the following:   * Graphical user interface (GUI)   + Knowledge of Swing components and containers   ***Goals:*** Reinforce your ability to create a GUI using Swing  ***Outcomes:*** You will master the following skills:   * Produce applications that use a Swing GUI  Background In this assignment, you will create a GUI that displays the Gourmet Coffee System's product catalog. Part of the work has been done for you and is provided in the student archive. You will complete the code that creates a graphical presentation of the product details. Description Class CatalogGUI lets the user display the product details of every product in the gourmet coffee store's product catalog. This simple GUI contains the following components:   * a JList that displays the product code of every product in the catalog * a JPanel that presents product details * a JTextArea that serves as a status area   To examine the details of a particular product, the user selects the product code in the list. The application responds by displaying the product details in the JPanel and a status message in the JTextArea.   |  | | --- | | Execution of CatalogGUI | | **Figure 1** *Execution of CatalogGUI* |   The following class diagram highlights the classes used to implement the GUI:   |  | | --- | | Portion of Gourmet Coffee System class diagram | | **Figure 2** *Portion of Gourmet Coffee System class diagram* |   Class CatalogGUI instantiates the Swing components, arranges the components in a window, and handles the events generated by the list. In this assignment, you will implement the method getDataFieldsPanel that returns a JPanel displaying the product details. An incomplete implementation of CatalogGUI is provided in the student archive.  Class DataField contains a name/value pair that represents a single attribute that is stored within an object. A complete implementation of DataField is provided in the student archive.  Classes Product, Coffee, and CoffeeBrewer have been enhanced to add a method called getDataFields that returns an ArrayList of DataField objects with the product details:   * In class Product, the method getDataFields returns an ArrayList with DataField objects for the attributes "code", "description" and "price". * In class Coffee, the method getDataFields returns an ArrayList with DataField objects for the attributes "code", "description", "price", "origin", "roast", "flavor", "aroma", "acidity" and "body". * In class CoffeeBrewer, the method getDataFields returns an ArrayList with DataField for the attributes "code", "description", "price", "model", "waterSupply" and "numberOfCups".   A complete implementation of classes Product, Coffee, and CoffeeBrewer are provided in the student archive.  Class Catalog has been enhanced: it now contains a method called getCodes that returns an array of product codes (all the product codes in the product catalog) which CatalogGUI uses to populate the JList. A complete implementation of Catalog is provided in the student archive. Files The following files are needed to complete this assignment:   * [*student-files.zip*](https://www.icarnegie.com/content/SSD/SSD3/4.2.0.0/normal/pg-adv-class-impl/pg-gui/assm-exer-gui/pool-pr-gui/qn-pr-bsc-gui-gou-cof/handout/student-files.zip) — Download this file. This archive contains the following:   + Class files     - *Product.class*     - *Coffee.class*     - *CoffeeBrewer.class*     - *Catalog.class*     - *CatalogLoader.class*     - File*CatalogLoader.class*     - *DataFormatException.class*     - *DataField.class*   + Documentation     - *Product.html*     - *Coffee.html*     - *CoffeeBrewer.html*     - *Catalog.html*     - *CatalogLoader.html*     - File*CatalogLoader.html*     - *DataFormatException.html*     - *DataField.html*   + Java files     - *CatalogGUI.java*— a complete implementation   + Data files for the test driver     - *catalog.dat* — a file with product information for every product in the product catalog  Tasks The following steps will guide you through this assignment. Work incrementally and test each increment. Follow Sun's code conventions. Save often.   1. **Extract** the files from student-files.zip 2. **Then**, complete the implementation of the method getDataFieldsPanel in the class CatalogGUI.   private JPanel getDataFieldsPanel(ArrayList<DataField> dataFields) — Returns a reference to a JPanel object that shows the names and values of the DataField objects stored in the parameter dataFields:   * + - For a coffee product, dataFields contains nine DataField objects with the names: "Code", "Description", "Price", "Origin", "Roast", "Flavor", "Aroma", "Acidity" and "Body".     - For a coffee brewer, dataFields contains six DataField objects with the names: "Code", "Description", "Price", "Model", "Water supply" and "Number of cups".     - For a generic product, dataFields contains three DataField objects with the names: "Code", "Description" and "Price".   For each DataField object stored in dataFields, the JPanel should contain a JLabel object with the name of the attribute and an uneditable JTextField object with the value of the attribute. Use the JPanel layering technique to build the JPanel. The arrangement of the JPanel need not match Figure 1 exactly; any well-organized presentation of the attributes will be acceptable.   1. **Finally**, compile and execute the class CatalogGUI.  Submission Upon completion, submit **only** the following.   1. CatalogGUI.java , CatalogGUI.class 2. a word file including the program running results |