How-To Guide for
Importing External Files into Eclipse

Overview

Learn how to import .txt (Adequate-Proficient) or .csv (Excellence) files into an Eclipse project, utilize them within your Java program, and store the contents into an appropriately typed ArrayList.

Requirements

1. Download and install the latest version of Eclipse IDE if you haven't already (https://www.eclipse.org/downloads/).
2. Locate data for your project. You can find sample data on websites like Kaggle (https://www.kaggle.com/datasets), City of Edmonton Open Data Portal (https://data.edmonton.ca/), or another site of your choice. Choose a dataset with .txt (for Adequate-Proficient) or .csv (for Excellence) format.
3. For this assignment, you will be required to create a new Java project in Eclipse and import external files (.txt or .csv) into the project.
4. After importing the files, write a Java program that reads and displays the content of these files. Store the contents of the .txt or .csv files into an appropriately typed ArrayList.
5. Use proper coding standards, including comments and indentation.

Steps in Guide

1. Create a new Java project in Eclipse IDE.
    - Open Eclipse and go to File > New > Java Project.

    - Name your project "ExternalFilesImport" and click Finish.
2. Create a package and class for your project.
    - Right-click on the "src" folder, go to New > Package, and name it "demo".
    - Inside the package, create a new class named "FileReaderDemo" with a main method.
3. Add .txt or .csv files to your project.
    - Create a folder named "data" in the project's root directory.
    - Place your .txt file (for Adequate-Proficient) or .csv file (for Excellence) into the "data" folder.
    - Refresh your project in Eclipse to see the new folder and files.
4. Write a Java program to read and display the contents of the .txt or .csv files, and store them into an appropriately typed ArrayList.
    - In the "FileReaderDemo" class, use the java.io and java.nio.file packages to read and display the contents of the .txt or .csv files.
    - Use try-catch blocks to handle any exceptions that may occur while reading the files.
    - Create an ArrayList to store the contents of the files, with the appropriate data type for the dataset you have chosen.
5. **Cleaning data** and handling blank cells.
    - While reading the data from the files, ensure that you handle blank cells appropriately.
    - Implement a method to clean the data before adding it to the ArrayList, following best practices for handling missing or blank data.
    - You can do this in Excel or Google Sheets if you wish.
6. Test your program by running it in Eclipse.
    - Right-click on the "FileReaderDemo" class in the Package Explorer, and click Run As > Java Application.
    - Verify that the contents of the .txt or .csv files are displayed correctly in the console, the data has been cleaned, and the ArrayList is populated with the data.
7. Once the program is working as expected, submit your Guide.

Grading

Your guide will be graded on its completeness, clarity, and attention to detail. Make sure to include all necessary steps and screenshots to make the process easy to follow. Your guide should also be well-organized and free from errors.

Resources

You are expected to use external resources to complete this assignment. Make sure to cite any sources you use in your guide.

- .txt: https://replit.com/@CorbettArtym/File-IO-20
- .csv: https://www.javatpoint.com/how-to-read-csv-file-in-java

Check out the below method examples!

Suggestion: by utilizing polymorphism, you can create methods with the same name that accept different type parameters. This allows your application to handle various data types while maintaining a consistent method naming convention. In addition, you will need to import some libraries to use some of these methods.

```
	// Data is read in one line at a time as a STRING. To create a 
INT, Double,
	// or Boolean method, you need to parse the String:
	// Boolean.parseBoolean(someString)
	private static ArrayList<String> loadStringList(String 
filename) {
		ArrayList<String> temp = new ArrayList<String>();
		try {
			BufferedReader file = new BufferedReader(new 
FileReader(filename));
			while (file.ready()) {
				temp.add(file.readLine());
			} // end while
			file.close();
		} catch (IOException e) {
			System.out.println(e);
		}

		return temp;

		// For non-String ArrayLists you need to PARSE the data 
before adding it
		// You might also want to trim it.

	}// end loadStringArr from a text file


	// Data is read in one line at a time as an Integer. To create 
a Double,
	// or Boolean method, you need to parse the String:
	// Boolean.parseBoolean(someString)
	private static ArrayList<Integer> loadIntegerList(String 
filename) {
		ArrayList<Integer> temp = new ArrayList<Integer>();
		try {
			BufferedReader file = new BufferedReader(new 
FileReader(filename));
			while (file.ready()) {
				String nextLineS = file.readLine();
```

```
                      try{
                                 int nextLineI =
Integer.parseInt(nextLineS);
                                 temp.add(nextLineI);
                      }catch (NumberFormatException ex){
            ex.printStackTrace();
        }

                 } // end while
                 file.close();
           } catch (IOException e) {
                 System.out.println(e);
           }

           return temp;

           // For non-String ArrayLists you need to PARSE the data
before adding it
           // You might also want to trim it.

      }// end loadIntegerArr from a text file


      // Data is read in one line at a time as a Double.
      // or Boolean method, you need to parse the String:
      // Boolean.parseBoolean(someString)
      private static ArrayList<Double> loadDoubleList(String
filename) {
           ArrayList<Double> temp = new ArrayList<Double>();
           try {
                 BufferedReader file = new BufferedReader(new
FileReader(filename));
                 while (file.ready()) {
                      String nextLineS = file.readLine();
                      try{
                                 double nextLineD =
Double.parseDouble(nextLineS);
                                 temp.add(nextLineD);
                      }catch (NumberFormatException ex){
            ex.printStackTrace();
        }

                 } // end while
                 file.close();
           } catch (IOException e) {
                 System.out.println(e);
           }

           return temp;

      }// end loadDoubleList from a text file
```

```
//Class Start
     //ArrayLists as global variables!!



     //main method start
          //call import method

     //main method end

     //import method start

     //import method end



//Class End
```

Name(s): ______________________________ **Block**: 1 2 3 4

How-To Guide for Importing External Files into Eclipse Rubric

Exemplary	**Proficient**	**Adequate**	**Limited**
☐ The guide includes all necessary steps and screenshots to make the process easy to follow. ☐ The guide is well-organized and free from errors. ☐ The guide is clear and concise, using appropriate language for the target audience. ☐ The guide includes proper citations for external resources used. ☐ The guide goes above and beyond, including additional helpful tips and tricks. ☐ .csv is required for Excellence!	☐ The guide includes most necessary steps and screenshots, but may be missing some minor details. ☐ The guide is generally well-organized and free from major errors. ☐ The guide is clear and uses appropriate language for the target audience. ☐ The guide includes citations for external resources used. ☐ The guide meets the basic requirements, but could benefit from additional details or explanations.	☐ The guide includes some necessary steps and screenshots, but may be missing important details. ☐ The guide may have some organizational or grammatical errors, but is generally understandable. ☐ The guide includes some citations for external resources used. ☐ The guide meets the minimum requirements, but lacks clarity or completeness.	☐ The guide is incomplete or missing necessary steps and screenshots. ☐ The guide has significant organizational or grammatical errors, making it difficult to understand. ☐ The guide does not include proper citations for external resources used. ☐ The guide does not meet the minimum requirements and may not be usable as a guide.

* Similar to AP rubrics, you must correctly have everything in the rightmost columns to score in high categories.