Advanced Java

LESSON 3: FILE ALGORITHMS AND APPLICATION

Creative Commons

Attribution 4.0 International (CC BY 4.0)

This workforce product was funded by a grant awarded by the U.S. Department of Labor's Employment and Training Administration. The product was created by the grantee and does not necessarily reflect the official position of the U.S. Department of Labor. The U.S. Department of Labor makes no guarantees, warranties, or assurances of any kind, express or implied, with respect to such information, including any information on linked sites and including, but not limited to, accuracy of the information or its completeness, timeliness, usefulness, adequacy, continued availability, or ownership. This is an equal opportunity program. Assistive technologies are available upon request and include Voice/TTY (771 or 800-947-6644).

Agenda

- File to Array
- Two File Match
- Control-Break Logic

File to Array

Overview

For this section, we are looking at problems that include using data from a file for reference, the data will not change.

- We only know Arrays and ArrayLists. In a few weeks, we will explore Collections which will provide several great solutions to common problems.
- A Database should be used if the file has a significant amount of data.
- The file may be Delimited, XML or JSON.
- Download, “FileIOAlgorithm” from Blackboard.

Entity Object

- The program we will explore uses Entity Objects.
- Most people think of Entity Objects as being objects that are instantiated from classes whose properties match the attributes of a table in a database. Or the columns from a select statement.
- The program that we will explore includes a class, **“Meals”**, that matches the lines in the file, **“meals_data.csv”**.
- This satisfies the definition of an Entity Object. Entity objects encapsulate the business policy and data for the logical structure of the business, such as product lines, departments, sales, and regions.

Entity Object

- File

```
Breakfast,Belgian waffle,647
```

- Class

```
public class Meals {
    private MealType mealType;
    private String item;
    private int calories;
```

Dealing with a CSV

- **split()**

```
String line;
String[] fields;
int i=0;
Meals[] meals = new Meals[100];
FileInput indata = new FileInput("meals_data.csv");
while ((line = indata.fileReadLine()) != null) {
    fields = line.split(",");
    meals[i++] = new
                 Meals(fields[0],fields[1],fields[2]);
}
```

Searching

- Enum

```
public void printMealsByType(MealType mealType) {
    for (Meals item: meals) {
        if (item != null && item.getMealType() == mealType) {
            System.out.println(item);
        }
    }
}
```

- String

```
public void printByNameSearch(String s) {
    for (Meals item: meals) {
        if (item != null && item.getItem().indexOf(s) >= 0) {
            System.out.println(item);
        }
    }
}
```

Dealing with Array Size

- This issue makes **`ArrayLists`** very attractive
- Ensure that no elements are added after the array has been filled

```
if (i < meals.length) {
    .  .  .
    meals[i++] = new Meals(mealType, arg2, calories);
}
else {
    System.out.println("Index Error");
}
```

Dealing with Array Size

- While searching or listing an array that is not completely filled, there will be elements with a **`null`** value.
- There are two solutions:
  - Use an if to verify that an element with a **`null`** value is not used.
  - Use a counter in the class that holds the Array to indicate the number of elements with valid values.

In-Class Activity

1. Download “FileIOAlgorithm” from Blackboard and open in IntelliJ.
2. Replace the Arrays with ArrayLists.

Two File Match

Overview

For this section, we are looking at algorithms for Account and Transaction files.

- One of the files needs to be identified as the Driver file. There is a primary loop. The other file is for additional info and the program needs to be able to continue regardless of the existance of matches.

Driver File

```
private final static FileInput cardAccts = new
    FileInput("movie_cards.csv");
private final static FileInput cardPur = new
    FileInput("movie_purchases.csv");
String line;String[] fields;
int[] nums = new int[2];
while ((line = cardAccts.fileReadLine()) != null) {
    fields = line.split(",");
    findPurchases(fields[0], nums);
    System.out.format("%6s  %-18s  %2d   %4d\n",
              fields[0],fields[1], nums[0], nums[1]);
}
```

Matching File

```
public static void findPurchases(String acct, int[] nums) {
nums[0] = 0;
nums[1] = 0;
String line;
String[] fields;
boolean done = false;
while (((line = cardPur.fileReadLine()) != null) && !(done)) {
    fields = line.split(",");
    if (fields[0].compareTo(acct) > 0) {
        done = true;
    }
    else if (fields[0].equals(acct)) {
        nums[0]++;
        nums[1] += Integer.parseInt(fields[2]);
    }
  }
}
```

In-Class Activity

1. Download “FileIOAlgorithmTwo” from Blackboard and open in IntelliJ.
2. Add a third file, “movie_rating.csv”.
3. This file includes two columns: Account Number and movie Rating.

```
100200,3
```

4. Add an average rating column to the output.

```
 Account   Name                 Movies Points Avg Rating
00100200   Savanah Costa           5      35      12.25
```

Control-Break Logic

Overview

For this section, we are looking at algorithms for subtotaling a sorted file.

- A Control Break is a change in the value of one of the keys on which a file is sorted which requires some extra processing.

Break Value

The field that the data is grouped by needs to be compared to a field that stores the previous value.

- Count and total fields need to be set to zero at the beginning and after each break.
- At each break, the subtotal is printed out.
- Ensure that the first input line is not processed as a break.

Break Code

```
String oldMealType="NA";
int total = 0;
int count = 0;
while ((line = meals.fileReadLine()) != null) {
   fields = line.split(",");
   if (!oldMealType.equals(fields[0])) {
      if (!oldMealType.equals("NA")) {
         System.out.print(oldMealType + " " + total);
      }
      oldMealType=fields[0];
      count = 0;
      total = 0;
   }
   total += Integer.parseInt(fields[2]);
   count++;
}
System.out.print(oldMealType + " " + total);
```

In-Class Activity

1. Modify “FileIOAlgorithmTwo” to create a CSV output file that includes a fields that are printed out.
2. Sort file by Movie Ratings – Look it up several ways to get it done.
3. Open file for input and print a line for each Movie Rating, 1-5, and how many ratings included that number.

```
Rating Count
   1       22
   2       41
   3       64
```