Reading a CSV file in Java

Reading a .CSV (comma separated volume) file in Java is a pretty straightforward way to use datasets in computer science. We are going to make a few classes to demonstrate how this is done. I am using a CSV file of Broadway shows for this example. The data and files for this are linked in the github repository for this project. A Runner is used to start our project as always. We model our Broadway data in its own class as well. The Controller is our tool we use to go between the data model and the user. Finally we have a DataReader for reading the data from the file.

Create your Java project in your IDE of choice. I am using Eclipse, but this will work with anything, even drjava.

Runner

Make the Runner class and put in an empty public static void main method.

Model class

Each column of the csv file is data we need and will need a corresponding data member in the model class. As always, data members are private! I follow the order of the columns in the CSV file for writing the declaration statements and constructor parameters to make it easy.

I recommend making a parameterized constructor and getter methods for each of the data members. We do not need setters in this class since we don't want to alter the values of our data points. I also want a toString method so we can describe each object as well. When making a toString I expect a description of the data using at least a few of the data members.

DataReader

We want to read in a data set so we can analyze the data and generate interesting information. We will need a method that can read a file and send the results back as an ArrayList of the type we are analyzing. We also need a method to handle any error that occurs if the file is unable to open. Since both these methods do not require internal information we are declaring them static and they will be called by ClassName.methodName instead of making an instance of the class and then calling the method on the instance.

Imports

I start with the imports I need to work with the data. To use a file we need java.io.File and java.io.FileNotFoundException if there are errors. To parse the text of the file we need a Scanner from java.util.Scanner. We are storing the results of the file in an ArrayList<Broadway> so we need java.util.ArrayList. Since this is a demo, all our classes are in the default package so no other import is needed.

Handling errors

Because opening a file may fail if there is a typo or the data is not readable we are going to make a method that can be called to let us know why the program was unable to open the file. This method is just going to display a message so it will be a void type. We will send the method an Exception parameter. As we are only calling it from a static method it will also need to be declared static. If it were not in a demo we could allow the user to respecify the file path information after.

Reading data

This is a public static method named readData that has a String parameter for the file path. This method needs to read the contents of the file and return an ArrayList of the Broadway data that it represents. It needs a parameter to specify the location of the file on our computer.

Since it is an ArrayList return type method we need to initialize the variable we are going to return when the method is finished to start and return it at the bottom of the method. Then we initialize a File object from the path parameter.
Next our method needs to try and load the information. We use a try with resources block to open the file using a Scanner instance that will close when the try block ends. The catch block has a FileNotFound exception parameter since that is the Exception type that is thrown if the Scanner is unable to read the supplied parameter.

Iterate over the file

The CSV file stores data by rows of text. Since we do not know how many rows exist in the file before opening it, we need to use the dynamic data structure ArrayList<String> to collect each row of data. Again, since we don't know how many there are, we will use a while loop based on the scanner's .hasNextLine call to read all lines from the file. Each line of the file is read as a String and added to the local ArrayList<String> variable we just initialized.

Once the lines are read we need to parse out the data for the data type we are working with. In this demo it is the Broadway information.

To iterate over the lines we start at index 1, since the first row of the CSV file contains the headers and not the values we are analyzing. In the test for the loop we check that the index is less than the size of the ArrayList of String and then increment to get each subsequent value.

Parse each line

We need to separate each String line into the data points of our model class. The split method in String supports this. It returns an array of String with each value from the line in its own cell.

Since the sample CSV file has 8 columns the .split method will return an array of the same length with the last index of 7. If a value is a primitive like an int,double, or boolean we need to use the Integer.parseInt, Double.parseDouble, or Boolean.parseBoolean methods appropriately to correctly convert from the String source into the other type.
I am making a local variable in the loop for each field and storing the parsed value from the String array of each line in the appropriate variable. Then we send those variables to the model constructor in the same order to initialize an instance of the object. Then we add that instance to the ArrayList<Broadway> we initialized at the start of the method. Finally the list is returned at the end of the method.

Using the data

In our analysis file, Controller.java we are going to test that the file loaded properly and that we can extract data from an instance

To make it easy, put the CSV file in the same directory as your Java project. You can practice using more complex path information another time. This makes loading the file VERY easy.

Demo method

Make a public void method called demoDataLoad that has no parameters. Inside the method call the DataReader.readData method with the name of the file as the parameter to initialize an ArrayList of the model data type. Use a System.out.println statement to verify the size of the ArrayList. Make a loop to print the contents from five indices so we can see different values.

Start the project

Create an instance of the controller in the PSVM of the Runner. And call the demoDataLoad method on it.

Continue

You can now write the methods you need to analyze the data to find out interesting information!