



The Company Employee Program

by Sophia



WHAT'S COVERED

In this lesson, you will construct a program by manipulating files using libraries and file inputs when using Java. Specifically, this lesson covers:

1. Storing Information to a File
2. Delete From File
3. Adding Main Menu



BEFORE YOU START

This tutorial utilizes the GDB IDE. If you started this course using the Replit IDE, please refer to the Replit [The Company Employee Program](#) tutorial to continue with directions, screenshots, and usage of the that IDE.

1. Storing Information to a File

In this lesson, we will give the employee class program some final touches! You will begin by creating a program that stores the employee ID, salary, last name, and first name for our employees into a file. Rather than having it entered every single time, you will write that information to a file so that we can easily recall that information. You could extend this program to include in the additional details of the employee, but instead, you will focus on the key elements (namely employee ID and salary). Let's first start by creating an *employees.csv* file by clicking on the 'New File' icon. Name it *employees.csv*.



TRY IT

Directions: Create a new file called *employees.csv*.



DID YOU KNOW

The new file is a spreadsheet file that has a suffix of csv. What is that? Comma separated value (or .csv) is a file format where each of the data elements for a row of data would be split by a comma. This is very common to spreadsheet applications like Microsoft Excel. In a .csv file, we would see data that is saved in a format like this example (it's important to have a blank line at the end of the file):

EXAMPLE

```
10001,75000,Johnson,Mary
```

10002,68500,Doe,John

This would be a file that contains the employee ID, salary, first name, and last name. As more information (individuals) is added, the data separation would continue.

Later, you will review the *employees.csv* file to see what that looks like as data is added to it. For now, the file should be completely empty.

As this is meant to be a larger program, start by defining a `CompanyEmployee` class to encapsulate data about an employee.



Directions: Add the following line of text to the `employee.csv` file:

10001,75000,Johnson,Mary

Then type the following code into a file named `CompanyEmployee.java`:

```
public class CompanyEmployee {
    private String lastName;
    private String firstName;
    int id;
    int salary;

    public CompanyEmployee(String lastName, String firstName, int id, int salary) {
        this.lastName = lastName;
        this.firstName = firstName;
        this.id = id;
        this.salary = salary;
    }

    public String getLastName() {
        return lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public int getId() {
        return id;
    }

    public int getSalary() {
        return salary;
    }
}
```

```
// toString() method provides format for printing company employee data:
// Smith, John ID: 10123 ($85000)
public String toString() {
    return lastName + ", " + firstName + " ID: " + id + " ($" + salary + ")";
}
}
```

Next, make a driver class for the application in a file named `CompanyEmployeesProgram.java`. We will start with an empty `main()` method.



Directions: Type in this code to start with, in a file named `CompanyEmployeesProgram.java`:

```
class CompanyEmployeesProgram {
    public static void main(String[] args) {

    }
}
```

You will return to the code in `main()` later, but let's continue at this point with a method to read the data from the `.csv` file and convert it into an `ArrayList` of `CompanyEmployee` objects. You will call the method to read the data `readEmployees()`. Since it will be part of the driver class and called from `main()`, it will need to be declared as a static method. This method takes a `String` parameter for passing in the name of the file. When done processing the data in the file, it will return an `ArrayList<CompanyEmployee>` collection. The line with the return type and the signature for the method should look like this:

↪ EXAMPLE

```
public static ArrayList readEmployees(String csvFile) {
```

Since the code will work with an `ArrayList`, the code will need to import `java.util.ArrayList`. Since we will be working with files, we also need the `File` and `Files` classes (along with associated exception types). We might as well go ahead and add the other needed import statements at the top of the file. The complete list reads:

↪ EXAMPLE

```
import java.util.ArrayList; import java.io.File; import java.io.FileNotFoundException; import java.io.IOException;
import java.nio.file.Files;
```

This list could be simplified a bit using a wildcard (*), if you prefer:

↪ EXAMPLE

```
import java.util.ArrayList; import java.io.*; import java.nio.file.Files;
```

As part of the code statement of the `readEmployees()` *method*, we create an empty `ArrayList` of `CompanyEmployee` objects called `employeeList`. Then, we open up the data file using a `File` object and use a `Scanner` to read the data from the file. The code then takes a line from the file and splits the data into individual data elements (an array of `String` data).

The data is read from the file as `String` data, but the first two items (the employee ID and the salary) need to be converted to integer values. The `Integer` wrapper class that we met when discussing generic types includes a method named `Integer.parseInt()`, which takes a `String` value and converts it to an `int` value.

Here is a partial initial draft of the method:

```
public static ArrayList<CompanyEmployee> readEmployees(String csvFile) {
    // Create an empty ArrayList of CompanyEmployee objects
    ArrayList<CompanyEmployee> employeeList = new ArrayList<>();

    // File object for accessing the CSV file
    File inputDataFile = new File(csvFile);
    List<String> lines = new ArrayList<>();
    // Because the following statements can through exceptions, they are in a try block
    try {
        lines = Files.readAllLines(inputDataFile.toPath());
        for(String line : lines) {
            String[] employeeData = line.split(",");
            int id = Integer.parseInt(employeeData[0]);
            int salary = Integer.parseInt(employeeData[1]);
            // Last name & first name don't need conversion to another datatype.
            String lastName = employeeData[2];
            String firstName = employeeData[3];
            // Now construct a CompanyEmployee object for each employee
            CompanyEmployee empl = new CompanyEmployee(lastName, firstName, id, salary);
            // Add CompanyEmployee object to ArrayList
            employeeList.add(empl);
        }
    }
    catch(FileNotFoundException ex) {
        System.out.println("File not found: " + ex.getMessage());
    }
    catch(IOException ex) {
        System.out.println("I/O error: " + ex.getMessage());
    }
    catch(NumberFormatException ex) {
        System.out.println("Number Format Error: " + ex.getMessage());
    }
    return employeeList;
}
```



Directions: Add the `readEmployees()` method to your `CompanyEmployeesProgram.java`. Don't forget the import statements at the top of the file.

Here is the first line of data from the `.csv` file:

↪ EXAMPLE

```
10001,75000,Johnson,Mary
```



REFLECT

This CSV data shows an employee ID of 10001, a salary of \$75000, a last name of Johnson, and a first name of Mary. Each line in the file corresponds to another employee. However, there are some potential errors and problems that could exist. Let's give it a try by switching the name slightly for the file to *wrong.csv* file instead of *employees.csv* file.



TRY IT

Directions: Add this code for the `main()` method:

```
public static void main(String[] args) {  
    // Try using the wrong file name  
    ArrayList<CompanyEmployee> employees = readEmployees("wrong.csv");  
}
```



TRY IT

Directions: Next, switch out the filename for *wrong.csv* and try running the program.



REFLECT

Do you see the same output as shown below?

```
I/O error: wrong.csv
```

This output is produced by the second catch block (rather than the rather messy output from an uncaught exception).



TRY IT

Directions: Change the file name back to the correct name `employees.csv` before continuing. The second catch block handles `NumberFormatException`. This type of exception occurs if the `Integer.parseInt()` can't convert a `String` to an `int` because it contains non-numeric characters. Open the `employees.csv` file and edit the first line so that the ID begins with the letter A (which doesn't occur in a base 10 integer) like this:

↪ EXAMPLE

```
A0001,75000,Johnson,Mary
```

Running the program now triggers the third catch block, so the output looks like this:

Number Format Error: For input string: "A0001"



Directions: Edit the first line in the CSV file to correct the entry so that it starts with 1 rather than A:

↪ EXAMPLE

```
10001,75000,Johnson,Mary
```



Directions: Go ahead and run the code when the file name and the data are correct. No exceptions should be thrown, so now there should be no output from the program:



There is no output as there were no errors. The program up to this point shouldn't have any errors to begin with. Now you can move on to the next step to be able to write to the file:

```
public static void writeEmployees(String csvFile, ArrayList<String> employees) {  
    File outputFile = new File(csvFile);  
    try {  
        // Write to output file in APPEND mode  
        Files.write(outputFile.toPath(), employees, StandardOpenOption.APPEND);  
    }  
    catch(IOException ex) {  
        System.out.println("Error writing to file: " + ex.getMessage());  
    }  
}
```



In this `writeEmployees()` method, you have the parameters set to take the `employees` list. Similar to the `readEmployees()` method, you will also incorporate the `try` and `catch` blocks since you are setting this method up for error handling as well. Within the `try` block, you are opening up the `employees.csv` file and writing to it in append mode, thanks to the argument `StandardOpenOption.APPEND`.

Here is an example of how writing to the file is handled by this line in the method:

↪ EXAMPLE

```
Files.write(outputFile.toPath(), employees, StandardOpenOption.APPEND);
```



The first argument is provided by the call to `OutputFile.toPath()` that converts the `File` object to the `Path` object that the `Files.write()` method expects. The second argument is the iterable `ArrayList` with the lines for the employees to write to the file. The last argument, `StandardOpenOption.APPEND`, puts the `write()` method into append mode.



TRY IT

Directions: Next, add the `writeEmployees()` method to your program:

Let's take the next step to create a method that prompts the user for the new employee's information.



TRY IT

Directions: We will name this `addEmployees()` and pass in a `Scanner` to read from the console the name of the CSV file for the output:

```
public static void addEmployees(Scanner input, String csvFile) {
    ArrayList<CompanyEmployee> employeesToAdd = new ArrayList<>();
    char keepGoing = 'Y';
    while(keepGoing == 'Y') {
        System.out.print("Enter Employee Last Name: ");
        String last = input.nextLine();
        System.out.print("Enter Employee First Name: ");
        String first = input.nextLine();
        System.out.print("Enter Employee ID#: ");
        int id = input.nextInt();
        System.out.print("Enter Salary: ");
        int salary = input.nextInt();
        // Remove new line remaining in input buffer
        input.nextLine();
        CompanyEmployee employee = new CompanyEmployee(last, first, id, salary);
        employeesToAdd.add(employee);

        // Check if user want to continue adding employees
        System.out.print("Continue adding? (Y/N): ");
        // Read line as a String but just grab 1st char
        keepGoing = input.nextLine().charAt(0);
    }

    // Now call method to write new data to file
    writeEmployees("employee.csv", employeesToAdd);
}
```



REFLECT

After reading in the data entered by the user, we create a `CompanyEmployee` object and add it to the `ArrayList` of employees to add to the file.



BIG IDEA

You may have noticed that we have been creating variables that are very similar, `employee` and `employees`, inside our functions. As long as these variables are within local scope, we will have no issues. Remember, if we were to pull one or two of these out from the function scope, we could potentially have scope errors like naming collisions. Always keep in mind that when creating variables, you should know what they are called and where they exist.

This newly created variable `employee` is appended to the `employees` list that was passed into the function. After this is done, we call the `write_employees()` function passing in the `employees` list. After that has been returned, we output to the screen that the employee was added:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class CompanyEmployeesProgram {
    public static void main(String[] args) {
        ArrayList<CompanyEmployee> employees = readEmployees("employees.csv");
        for(CompanyEmployee empl : employees) {
            System.out.println(empl);
        }
        ArrayList<String> newEmployees = new ArrayList<>();
    }

    public static ArrayList<CompanyEmployee> readEmployees(String csvFile) {
        // Create an empty ArrayList of CompanyEmployee objects
        ArrayList<CompanyEmployee> employeeList = new ArrayList<>();

        // File object for accessing the CSV file
        File inputDataFile = new File(csvFile);
        List<String> lines = new ArrayList<>();
        // Because the following statements can through exceptions, they are in a try block
        try {
            lines = Files.readAllLines(inputDataFile.toPath());
            for(String line : lines) {
                String[] employeeData = line.split(",");
                int id = Integer.parseInt(employeeData[0]);
                int salary = Integer.parseInt(employeeData[1]);
                // Last name & first name don't need conversion to another datatype.
                String lastName = employeeData[2];
                String firstName = employeeData[3];
                // Now construct a CompanyEmployee object for each employee
            }
        }
    }
}
```



```

        CompanyEmployee empl = new CompanyEmployee(lastName, firstName, id, salary);
        // Add CompanyEmployee object to ArrayList
        employeeList.add(empl);
    }
}

catch(FileNotFoundException ex) {
    System.out.println("File not found: " + ex.getMessage());
}

catch(IOException ex) {
    System.out.println("I/O error: " + ex.getMessage());
}

catch(NumberFormatException ex) {
    System.out.println("Number Format Error: " + ex.getMessage());
}

return employeeList;
}

public static void addEmployees(Scanner input, String csvFile) {
    ArrayList<CompanyEmployee> employeesToAdd = new ArrayList<>();
    char keepGoing = 'Y';
    while(keepGoing == 'Y') {
        System.out.print("Enter Employee Last Name: ");
        String last = input.nextLine();
        System.out.print("Enter Employee First Name: ");
        String first = input.nextLine();
        System.out.print("Enter Employee ID#: ");
        int id = input.nextInt();
        System.out.print("Enter Salary: ");
        int salary = input.nextInt();
        // Remove new line remaining in input buffer
        input.nextLine();
        CompanyEmployee employee = new CompanyEmployee(last, first, id, salary);
        employeesToAdd.add(employee);

        // Check if user want to continue adding employees
        System.out.print("Continue adding? (Y/N): ");
        // Read line as a String but just grab 1st char
        keepGoing = input.nextLine().charAt(0);
    }

    // Now call method to write new data to file
    writeEmployees("employee.csv", employeesToAdd);
}

public static void writeEmployees(String csvFile, ArrayList<CompanyEmployee> employees) {
    // Convert ArrayList<CompanyEmployee> to ArrayList<String>

```

```

ArrayList<String> newEmployees = new ArrayList<>();
for(CompanyEmployee empl : employees) {
    newEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
        "," + empl.getFirstName());
}
File outputFile = new File(csvFile);
try {
    // Write to output file in APPEND mode
    Files.write(outputFile.toPath(), newEmployees, StandardOpenOption.APPEND);
}
catch(IOException ex) {
    System.out.println("Error writing to file: " + ex.getMessage());
}
}
}

```

Let's put this together with what we have so far by first returning the employees list from `read_employees()` function, and then we will call the `add_employee()` function.



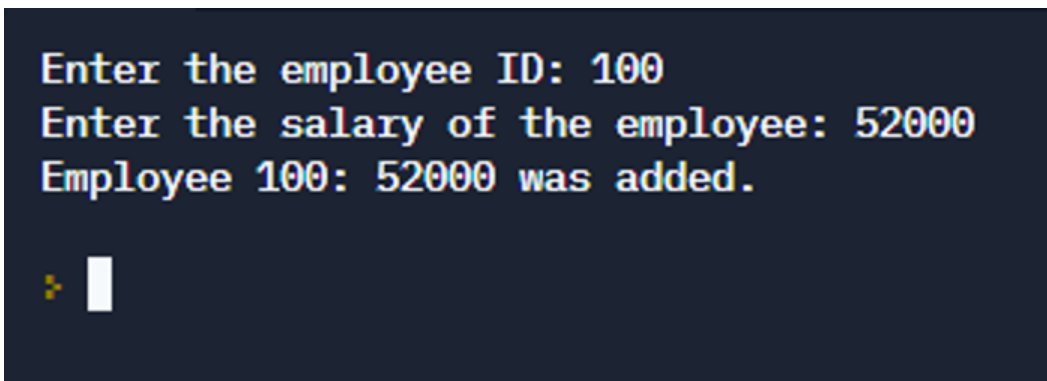
TRY IT

Directions: Add the `add_employee()` function to your program. Remember to also add the calls to the `read_employees()` and `add_employee()` functions at the bottom. When finished, run the program and give input for the employee ID and the salary of your first employee.

↪ EXAMPLE

Use 100 as ID and 52,000 as salary.

The output screen would reflect:



On screen, it looks correct so far. Let's take a look at what is in the `employees.csv` file.



TRY IT

Directions: See if your `employee.csv` file is showing the correct data too.

Here are the expected results:

```
employees.csv ×
1 100,52000
2
```

 REFLECT

This looks good, as well as the data was saved correctly.

 TRY IT

Directions: Run the program again now.

We expect to see the second employee ID as 101 and salary of 25000.

```
Enter the employee ID: 101
Enter the salary of the employee: 25000
Employee 101: 25000 was added.

>
```

Looking back again at the `employees.csv` file, we should see:

```
employees.csv ×
1 100,52000
2 101,25000
3
```

 REFLECT

This is useful, but it's probably not always ideal to have to keep looking at the `employees.csv` file. As an alternative, you could create a function to output the list of employees. Remember back when you first learned about loops, that an iterable is any object that can return its members one at a time. The `enumerate()` function from Java allows you to loop in the same way with the data.

Here is a method called `listEmployees()` that prints out a numbered list based on the data in the CSV file (which is passed in as a parameter):

```
public static void listEmployees(String csvFile) {
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    int menuNumber = 1;
    for(CompanyEmployee empl : employees) {
        System.out.println(menuNumber++ + ". " + empl);
    }
}
```



REFLECT

Let's break down this method. Here, you are passing the name of the file as a string. The contents of the CSV file are read into an `ArrayList`, which is, in turn, iterated over by an enhanced `for` loop. The variable `menuNumber` is initialized to 1 (since the menu numbers need to count the way people do, not starting with 0 as the computer does) and incremented on each pass through the loop. The `println()` relies on the `CompanyEmployee` class's `toString()` method (implicitly, since it is not called directly) to format the output so that it is simple to read. In this line, note how the `++` increment operator is placed right after the variable name and is then followed by a space before the `+` that does the concatenation:

⇒ EXAMPLE

```
System.out.println(menuNumber++ + ". " + empl);
```

The current value of `menuNumber` is concatenated first and then incremented (so the new value will be in place for the next iteration).

Let's review what happens when we run this function with the rest of our program.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class CompanyEmployeesProgram {
    public static void main(String[] args) {
        listEmployees("employees.csv");
    }

    public static ArrayList<CompanyEmployee> readEmployees(String csvFile) {
        // Create an empty ArrayList of CompanyEmployee objects
```

```

ArrayList<CompanyEmployee> employeeList = new ArrayList<>();

// File object for accessing the CSV file
File inputDataFile = new File(csvFile);
List<String> lines = new ArrayList<>();
// Because the following statements can through exceptions, they are in a try block
try {
    lines = Files.readAllLines(inputDataFile.toPath());
    for(String line : lines) {
        String[] employeeData = line.split(",");
        int id = Integer.parseInt(employeeData[0]);
        int salary = Integer.parseInt(employeeData[1]);
        // Last name & first name don't need conversion to another datatype.
        String lastName = employeeData[2];
        String firstName = employeeData[3];
        // Now construct a CompanyEmployee object for each employee
        CompanyEmployee empl = new CompanyEmployee(lastName, firstName, id, salary);
        // Add CompanyEmployee obejct to ArrayList
        employeeList.add(empl);
    }
}
catch(FileNotFoundException ex) {
    System.out.println("File not found: " + ex.getMessage());
}
catch(IOException ex) {
    System.out.println("I/O error: " + ex.getMessage());
}
catch(NumberFormatException ex) {
    System.out.println("Number Format Error: " + ex.getMessage());
}
return employeeList;
}

public static void addEmployees(Scanner input, String csvFile) {
    ArrayList<CompanyEmployee> employeesToAdd = new ArrayList<>();
    char keepGoing = 'Y';
    while(keepGoing == 'Y') {
        System.out.print("Enter Employee Last Name: ");
        String last = input.nextLine();
        System.out.print("Enter Employee First Name: ");
        String first = input.nextLine();
        System.out.print("Enter Employee ID#: ");
        int id = input.nextInt();
        System.out.print("Enter Salary: ");
        int salary = input.nextInt();
        // Remove new line remaining in input buffer

```

```

        input.nextLine();

        CompanyEmployee employee = new CompanyEmployee(last, first, id, salary);
        employeesToAdd.add(employee);

        // Check if user want to continue adding employees
        System.out.print("Continue adding? (Y/N): ");
        // Read line as a String but just grab 1st char
        keepGoing = input.nextLine().charAt(0);
    }

    // Now call method to write new data to file
    writeEmployees("employee.csv", employeesToAdd);
}

public static void writeEmployees(String csvFile, ArrayList<CompanyEmployee> employees) {
    // Convert ArrayList<CompanyEmployee> to ArrayList<String>
    ArrayList<String> newEmployees = new ArrayList<>();
    for (CompanyEmployee empl : employees) {
        newEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
            "," + empl.getFirstName());
    }
    File outputFile = new File(csvFile);
    try {
        // Write to output file in APPEND mode
        Files.write(outputFile.toPath(), newEmployees, StandardOpenOption.APPEND);
    }
    catch (IOException ex) {
        System.out.println("Error writing to file: " + ex.getMessage());
    }
}

public static void listEmployees(String csvFile) {
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    int menuNumber = 1;
    for (CompanyEmployee empl : employees) {
        System.out.println(menuNumber++ + ". " + empl);
    }
}
}

```



Directions: Add the `list_employees()` function to your program. Notice that we replaced the call to `add_employee()` function with the new `list_employees()` function. This time, the program will not have input, only listing the contents of

the employee.csv. Remember to keep the `read_employees()` and `list_employees()` functions at the bottom. When finished, run the program.

```
~/.../main/java$ java CompanyEmployeesProgram.java
```

```
1. Johnson, Mary ID: 10001 ($75000)
```

```
2. Doe, John ID: 10002 ($68500)
```

```
~/.../main/java$
```

The output is much cleaner now, as we have a centralized place to see the list of employees with the employee ID and the salary for each.

2. Delete From File

Next, you will create a method that will remove an employee from the file based on the employee ID.

```
public static boolean deleteEmployee(String csvFile, int emplID) {
    // Read contents of existing file
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    // Track if employee has been found and deleted
    boolean employeeDeleted = false;
    CompanyEmployee emplToDelete = null;
    for(CompanyEmployee empl : employees) {
        if(empl.getId() == emplID) {
            emplToDelete = empl;
        }
    }
    if(emplToDelete != null) {
        employees.remove(emplToDelete);
        employeeDeleted = true;
        ArrayList<String> remainingEmployees = new ArrayList<>();
        for(CompanyEmployee empl : employees) {
            remainingEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
                "," + empl.getFirstName());
        }
        File outputFile = new File(csvFile);
        try {
            // Write to output file - overwrite if it already exists
            Files.write(outputFile.toPath(), remainingEmployees, StandardOpenOption.TRUNCATE_EXISTING);
        }
        catch(IOException ex) {
            System.out.println("Error writing to file: " + ex.getMessage());
        }
    }
    return employeeDeleted;
}
```

```
}
```

In this method, we will prompt the user for an employee ID. Next, we will loop through the list to find that employee ID. If we find it in the list, we will remove it using the `remove()` method and set `employeeDeleted` to true. If, after looping through the entire list, the `employeeDeleted` variable is still false, it means that the employee was not found. Otherwise, the method writes an updated list with the item removed. Let's give it a try now.



Directions: Input the following to loop through the list:

```
import java.io.*;
import java.nio.file.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class CompanyEmployeesProgram {
    public static void main(String[] args) {
        System.out.println("Original List: ");
        listEmployees("employees.csv");
        System.out.println("Deleting employee...");
        deleteEmployee("employees.csv", 10002);
        System.out.println("Revised List: ");
        listEmployees("employees.csv");
    }

    public static ArrayList<CompanyEmployee> readEmployees(String csvFile) {
        // Create an empty ArrayList of CompanyEmployee objects
        ArrayList<CompanyEmployee> employeeList = new ArrayList<>();

        // File object for accessing the CSV file
        File inputDataFile = new File(csvFile);
        List<String> lines = new ArrayList<>();
        // Because the following statements can through exceptions, they are in a try block
        try {
            lines = Files.readAllLines(inputDataFile.toPath());
            for(String line : lines) {
                String[] employeeData = line.split(",");
                int id = Integer.parseInt(employeeData[0]);
                int salary = Integer.parseInt(employeeData[1]);
                // Last name & first name don't need conversion to another datatype.
                String lastName = employeeData[2];
                String firstName = employeeData[3];
                // Now construct a CompanyEmployee object for each employee
                CompanyEmployee empl = new CompanyEmployee(lastName, firstName, id, salary);
            }
        }
    }
}
```



```

        // Add CompanyEmployee object to ArrayList
        employeeList.add(empl);
    }
}
catch(FileNotFoundException ex) {
    System.out.println("File not found: " + ex.getMessage());
}
catch(IOException ex) {
    System.out.println("I/O error: " + ex.getMessage());
}
catch(NumberFormatException ex) {
    System.out.println("Number Format Error: " + ex.getMessage());
}
return employeeList;
}

public static void addEmployees(Scanner input, String csvFile) {
    ArrayList<CompanyEmployee> employeesToAdd = new ArrayList<>();
    char keepGoing = 'Y';
    while(keepGoing == 'Y') {
        System.out.print("Enter Employee Last Name: ");
        String last = input.nextLine();
        System.out.print("Enter Employee First Name: ");
        String first = input.nextLine();
        System.out.print("Enter Employee ID#: ");
        int id = input.nextInt();
        System.out.print("Enter Salary: ");
        int salary = input.nextInt();
        // Remove new line remaining in input buffer
        input.nextLine();
        CompanyEmployee employee = new CompanyEmployee(last, first, id, salary);
        employeesToAdd.add(employee);

        // Check if user want to continue adding employees
        System.out.print("Continue adding? (Y/N): ");
        // Read line as a String but just grab 1st char
        keepGoing = input.nextLine().charAt(0);
    }

    // Now call method to write new data to file
    writeEmployees("employee.csv", employeesToAdd);
}

public static void writeEmployees(String csvFile, ArrayList<CompanyEmployee> employees) {
    // Convert ArrayList<CompanyEmployee> to ArrayList<String>
    ArrayList<String> newEmployees = new ArrayList<>();

```

```

for(CompanyEmployee empl : employees) {
    newEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
        "," + empl.getFirstName());
}
File outputFile = new File(csvFile);
try {
    // Write to output file in APPEND mode
    Files.write(outputFile.toPath(), newEmployees, StandardOpenOption.APPEND);
}
catch(IOException ex) {
    System.out.println("Error writing to file: " + ex.getMessage());
}
}

public static void listEmployees(String csvFile) {
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    int menuNumber = 1;
    for(CompanyEmployee empl : employees) {
        System.out.println(menuNumber++ + ". " + empl);
    }
}

public static boolean deleteEmployee(String csvFile, int emplID) {
    // Read contents of existing file
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    // Track if employee has been found and deleted
    boolean employeeDeleted = false;
    CompanyEmployee emplToDelete = null;
    for(CompanyEmployee empl : employees) {
        if(empl.getId() == emplID) {
            emplToDelete = empl;
        }
    }
    if(emplToDelete != null) {
        employees.remove(emplToDelete);
        employeeDeleted = true;
        ArrayList<String> remainingEmployees = new ArrayList<>();
        for(CompanyEmployee empl : employees) {
            remainingEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
                "," + empl.getFirstName());
        }
        File outputFile = new File(csvFile);
        try {
            // Write to output file - overwrite if it already exists
            Files.write(outputFile.toPath(), remainingEmployees, StandardOpenOption.TRUNCATE_EXISTING);

```

```

    }
    catch(IOException ex) {
        System.out.println("Error writing to file: " + ex.getMessage());
    }
}
return employeeDeleted;
}
}

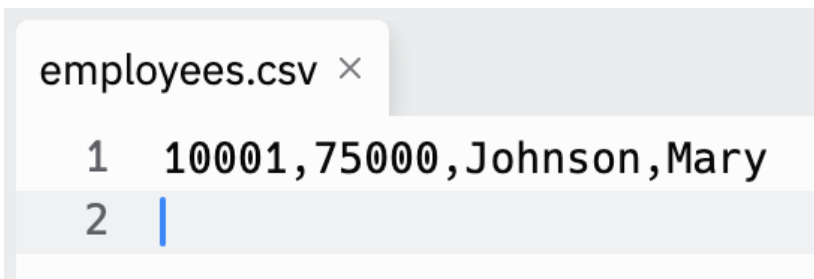
```

The output should look like the following:

```

~/.../main/java$ java CompanyEmployeesProgram.java
Original List:
1. Johnson, Mary ID: 10001 ($75000)
2. Doe, John ID: 10002 ($68500)
Deleting employee...
Revised List:
1. Johnson, Mary ID: 10001 ($75000)
~/.../main/java$
Looking at the employees file, the result is correct:

```



3. Adding Main Menu

Now that we have this much of the program written, next we will create a method and revise the code in the application's main() so that it runs more like a real application. You will set up a basic menu that will allow the user to know how to run the program to call each of these functions with a basic menu:

```

public static void displayMenu() {
    System.out.println("\nlist - List all employees");
    System.out.println(" add - Add an employee");
    System.out.println(" del - Delete an employee");
    System.out.println("exit - Exit program");
    System.out.print("Enter Command: ");
}

```

This method allows the user to enter list, add, del, or exit to work through the program. Now, we can set up a basic program that will loop through the program to display the user the menu, and allow the user to perform actions on our employee list using the functions we have created. Here is the revised main() method:

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String cmd = "";
    System.out.print("Enter the name of the CSV file: ");
    String csvFile = input.nextLine();

    while(!cmd.toLowerCase().equals("exit")) {
        displayMenu();
        cmd = input.nextLine();
        if(cmd.toLowerCase().equals("list")) {
            listEmployees(csvFile);
        }
        else if(cmd.toLowerCase().equals("add")) {
            addEmployees(input, csvFile);
        }
        else if(cmd.toLowerCase().equals("del")) {
            System.out.print("Employee ID to delete: ");
            int emplId = input.nextInt();
            deleteEmployee(csvFile, emplId);
        }
    }
}
```

The only difference here is that we're using a while loop to iterate until the exit is entered.

The completed program will look like the following:

```
import java.io.*;
import java.nio.file.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class CompanyEmployeesProgram {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String cmd = "";
        System.out.print("Enter the name of the CSV file: ");
        String csvFile = input.nextLine();

        while(!cmd.toLowerCase().equals("exit")) {
            displayMenu();
```

```

        cmd = input.nextLine();
        if(cmd.toLowerCase().equals("list")) {
            listEmployees(csvFile);
        }
        else if(cmd.toLowerCase().equals("add")) {
            addEmployees(input, csvFile);
        }
        else if(cmd.toLowerCase().equals("del")) {
            System.out.print("Employee ID to delete: ");
            int emplId = input.nextInt();
            // Remove remaining \n in input
            input.nextLine();
            deleteEmployee(csvFile, emplId);
        }
    }
}

public static void displayMenu() {
    System.out.println("\nlist - List all employees");
    System.out.println(" add - Add an employee");
    System.out.println(" del - Delete an employee");
    System.out.println("exit - Exit program");
    System.out.print("Enter Command: ");
}

public static ArrayList<CompanyEmployee> readEmployees(String csvFile) {
    // Create an empty ArrayList of CompanyEmployee objects
    ArrayList<CompanyEmployee> employeeList = new ArrayList<>();

    // File object for accessing the CSV file
    File inputDataFile = new File(csvFile);
    List<String> lines = new ArrayList<>();
    // Because the following statements can through exceptions, they are in a try block
    try {
        lines = Files.readAllLines(inputDataFile.toPath());
        for(String line : lines) {
            String[] employeeData = line.split(",");
            int id = Integer.parseInt(employeeData[0]);
            int salary = Integer.parseInt(employeeData[1]);
            // Last name & first name don't need conversion to another datatype.
            String lastName = employeeData[2];
            String firstName = employeeData[3];
            // Now construct a CompanyEmployee object for each employee
            CompanyEmployee empl = new CompanyEmployee(lastName, firstName, id, salary);
            // Add CompanyEmployee obejct to ArrayList
            employeeList.add(empl);
        }
    }
}

```

```

    }
}
catch(FileNotFoundException ex) {
    System.out.println("File not found: " + ex.getMessage());
}
catch(IOException ex) {
    System.out.println("I/O error: " + ex.getMessage());
}
catch(NumberFormatException ex) {
    System.out.println("Number Format Error: " + ex.getMessage());
}
return employeeList;
}

public static void addEmployees(Scanner input, String csvFile) {
    ArrayList<CompanyEmployee> employeesToAdd = new ArrayList<>();
    char keepGoing = 'Y';
    while(keepGoing == 'Y') {
        System.out.print("Enter Employee Last Name: ");
        String last = input.nextLine();
        System.out.print("Enter Employee First Name: ");
        String first = input.nextLine();
        System.out.print("Enter Employee ID#: ");
        int id = input.nextInt();
        System.out.print("Enter Salary: ");
        int salary = input.nextInt();
        // Remove new line remaining in input buffer
        input.nextLine();
        CompanyEmployee employee = new CompanyEmployee(last, first, id, salary);
        employeesToAdd.add(employee);

        // Check if user want to continue adding employees
        System.out.print("Continue adding? (Y/N): ");
        // Read line as a String but just grab 1st char
        keepGoing = input.nextLine().charAt(0);
    }

    // Now call method to write new data to file
    writeEmployees(csvFile, employeesToAdd);
}

public static void writeEmployees(String csvFile, ArrayList<CompanyEmployee> employees) {
    // Convert ArrayList<CompanyEmployee> to ArrayList<String>
    ArrayList<String> newEmployees = new ArrayList<>();
    for(CompanyEmployee empl : employees) {
        newEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +

```

```

        "," + empl.getFirstName());
    }
    File outputFile = new File(csvFile);
    try {
        // Write to output file in APPEND mode
        Files.write(outputFile.toPath(), newEmployees, StandardOpenOption.APPEND);
    }
    catch(IOException ex) {
        System.out.println("Error writing to file: " + ex.getMessage());
    }
}

public static void listEmployees(String csvFile) {
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    int menuNumber = 1;
    for(CompanyEmployee empl : employees) {
        System.out.println(menuNumber++ + ". " + empl);
    }
}

public static boolean deleteEmployee(String csvFile, int emplID) {
    // Read contents of existing file
    ArrayList<CompanyEmployee> employees = readEmployees(csvFile);
    // Track if employee has been found and deleted
    boolean employeeDeleted = false;
    CompanyEmployee emplToDelete = null;
    for(CompanyEmployee empl : employees) {
        if(empl.getId() == emplID) {
            emplToDelete = empl;
        }
    }
    if(emplToDelete != null) {
        employees.remove(emplToDelete);
        employeeDeleted = true;
        ArrayList<String> remainingEmployees = new ArrayList<>();
        for(CompanyEmployee empl : employees) {
            remainingEmployees.add(empl.getId() + "," + empl.getSalary() + "," + empl.getLastName() +
                "," + empl.getFirstName());
        }
        File outputFile = new File(csvFile);
        try {
            // Write to output file - overwrite if it already exists
            Files.write(outputFile.toPath(), remainingEmployees, StandardOpenOption.TRUNCATE_EXISTING);
        }
        catch(IOException ex) {

```

```

        System.out.println("Error writing to file: " + ex.getMessage());
    }
}
return employeeDeleted;
}
}

```



Directions: Try and see if the results from the employee is as expected:

The output should reflect the following:

```

~/.../main/java$ java CompanyEmployeesProgram.java
Enter the name of the CSV file: employees.csv

```

```

list - List all employees
add - Add an employee
del - Delete an employee
exit - Exit program
Enter Command: list
1. Johnson, Mary ID: 10001 ($75000)
2. Doe, John ID: 10002 ($68500)

```

```

list - List all employees
add - Add an employee
del - Delete an employee
exit - Exit program
Enter Command: add
Enter Employee Last Name: Jones
Enter Employee First Name: Ann
Enter Employee ID#: 12345
Enter Salary: 85000
Continue adding? (Y/N): N
list - List all employees
add - Add an employee
del - Delete an employee
exit - Exit program
Enter Command: list
1. Johnson, Mary ID: 10001 ($75000)
2. Doe, John ID: 10002 ($68500)
3. Jones, Ann ID: 12345 ($85000)

```

```

list - List all employees
add - Add an employee

```



```
del - Delete an employee
exit - Exit program
Enter Command: del
Employee ID to delete: 12345
```

```
list - List all employees
add - Add an employee
del - Delete an employee
exit - Exit program
Enter Command: list
1. Johnson, Mary ID: 10001 ($75000)
2. Doe, John ID: 10002 ($68500)
```

```
list - List all employees
add - Add an employee
del - Delete an employee
exit - Exit program
Enter Command: exit
~/.../main/java$
```



Everything looks accurate. Now the next time we launch the program, it should work exactly as expected with the employee list available to start with.

SUMMARY

In this lesson, you learned about how to create a program that allows the user to open a .csv file, add items to the .csv file, read from the .csv file and delete them. This included **storing information to a file, deleting from a file,** and **adding a main menu program** to help iterate through that entire process.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/